Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Implementació d'un sistema de signatura electrónica per web alternatiu als applets

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Albert Díaz Casellas

DIRECTOR: Olga León Abarca

SUPERVISOR: Román de Blas Ricart

DATA: 8 de juliol de 2016

Títol: Implementació d'un sistema de signatura electrónica per web alternatiu als applets

Autor: Albert Díaz Casellas

Director: Olga León Abarca

Data: 8 de juliol de 2016

Resum

La signatura digital, encara que lentament, està guanyant terreny davant la signatura convencional. A Espanya s'utilitza principalment a l'administració pública en processos de signatura de contractes de treball, i inclús algunes immobiliàries en els contractes de compra i venta d'habitacles. Cada cop hi ha més empreses de diferents àmbits que busquen la implementació de la signatura electrònica ja que permet agilitzar els tràmits i maximitzar el rendiment de les empreses.

D'altra banda, a l'any 2014 Google va decidir deixar de donar suport en el seu navegador Chrome a una tecnologia fins ara molt utilitzada anomenada *Netscape Plugin Application Programming Interface*, que permetia executar petits programes integrats en la web i realitzar tota mena d'operacions criptogràfiques. A més, sembla que aquesta decisió presa per Google és una tendència, ja que altres companyies han fet anuncis de que el seu navegador també desactivarà aquesta funcionalitat. Això ha comportat la necessitat de buscar noves tecnologies per substituir aquesta ja implementada.

En aquest projecte s'han avaluat diferents arquitectures amb els seus avantatges i inconvenients, i s'ha estudiat la solució més adient per a l'empresa Isigma, la qual treballa en el camp de la signatura digital. S'ha valorat el grau d'implementació que tindran les diferents solucions, l'impacte de les actualitzacions de *software*, tant dels diferents navegadors com dels sistemes operatius i de la tecnologia escollida per realitzar aquesta nova funcionalitat.

Aquesta solució s'ha implementat a l'aplicació web de la qual disposa Isigma, anomenada PortaSigma, i durant un temps es mantindrà una dualitat de tecnologies, perquè poc a poc, els usuaris vagin realitzant la transició.

Title: Implementation of a new digital signature system for web as an alternative to applets

Author: Albert Díaz Casellas

Director: Olga León Abarca

Date: July 8th, 2016

Overview

Digital signature, although slowly, is gaining ground over conventional signature. Mainly, it is used by the Spanish public administration, to sign employment contracts, and even it is being introduced to sign Real-State purchase agreements. It seems to be a trend because companies from different sectors are demanding this service more and more over time. This is because digital signature helps to speed up business performance.

On the other hand, in 2014 Google decided to withdraw its support in its browser to Netscape Plugin Application Programming Interface, a technology that was very used for executing little applications integrated into the webpages and allowed doing any kind of cryptographic operations. Besides, it seems this decision taken by Google has become a trend, and other companies are doing the same with their own browser. This situation has lead to the need to look for new technologies for replacing it.

In this project, we have evaluated differents architectures weighing up their advantages and disadvantadges, and haveistudied the best solution for Isigma, a company that offers digital signature services. In order to do that, we have taken into account their implementation viabilityand their impact on browser updates.

In particular, this architecture has been implemented in Portasigma, which is Isigma's web application, and both technologies will doexist for a while to allow a gradual transition of the users.

ÍNDEX

LLISTAT DE FIGURES

LLISTAT DE TAULES

ACRÒNIMS

AJAX: Asynchronous Javascript And XML
API: Application Programming Interface
CA: Certification Authority
CLSID: Class Identifier
CORS: Cross-Origin Resource Sharing
CS: ClickSign
CSS: Cascading Style Sheets
DLL: Dynamic-Link Library
DN: Distinguished Name
DRM: Digital Rights Management
EKU: Extended Key Usage
FTP: File Transfer Protocol
HTML5: HyperText Markup Language 5
HTTP: HyperText Transfer Protocol
HTTPS: HyperText Transfer Protocol Secure
IID: Interface IDentifier
ISM: Identity and Signature Managment
JAR: Java ARchive
JDK: Java Development Kit
JRE: Java Runtime Environment
JS: Javascript
JVM: Java Virtual Machine
KU: Key Usage
NPAPI: Netscape Plugin Application Programming Interface
NSIS: Nullsoft Scriptable Install System
NSS: Name Service Switch
OCSP: Online Certificate Status Protocol
PDF: Portable Document File
PE: Portable Execute
PKCS: Public-Key Cryptography Standards
POM: Project Object Model
PPAPI: PepperAPI
PS: Portasigma
SSL: Secure Socket Layer
TLS: Transport Security Layer
UI: User Interface
URL: Uniform Resource Locutor
UUID: Universally Unique Identifier
W10: Windows 10
W3C: World Wide Web Consortium
W7: Windows 7
XML: eXtensible Markup Language

INTRODUCCIÓ

Durant tota la història de la humanitat la signatura ha estat l'escriptura del nom i cognom de la persona escrits de la seva pròpia mà i uns traços que l'acompanyen. Aquesta escriptura realitzada pel signant és la que permet donar validesa a un document. Amb l'evolució de les tecnologies han aparegut altres mètodes, com la signatura digital, que també garanteixen la validesa, inclús altres propietats dels documents, com la seva integritat.

Poc a poc, aquesta nova forma de signar s'està integrant en les nostres vides. Per exemple, en el cas de realitzar tràmits online amb hisenda pública d'Espanya, o en empreses temporals de treball, que fan signar el contracte des d'una plataforma pròpia amb un *smartphone* o un ordinador. Aquesta tecnologia, principalment, s'està desenvolupant en llenguatge Java i executant-se en navegadors per a poder permetre la signatura en el *cloud*. D'aquesta forma el signant només ha d'entrar a la web que indiqui l'empresa i realitzar la signatura digital. I així, s'estalvien haver de realitzar tots els tràmits presencialment amb la conseqüent pèrdua de temps.

L'evolució de les tecnologies utilitzades a internet, com per exemple la versió 5 del llenguatge *HyperText Markup Language*, o l'ús del llenguatge Javascript, ha fet que grans empreses, com per exemple Google, estiguin prenent la decisió de no permetre en els seus navegadors web l'execució del Java i optant per aquestes noves tecnologies. Això comporta un greu problema en el camp dels certificats digitals i la criptografia, ja que el Java era l'única forma que hi havia de realitzar tota aquesta lògica.

L'objectiu d'aquest projecte és cercar tecnologies alternatives per substituir el Java, i integrar la solució més adient, valorant-la tant a nivell d'usuari com a nivell del desenvolupador, per a l'aplicació web Portasigma de l'empresa Isigma. Aquesta aplicació permet realitzar signatures en el *cloud*, això vol dir que, la persona que necessiti que el seu document sigui signat, digitalitzarà el document, el pujarà a Portasigma i associarà una petició de signatura al DNI de la persona corresponent i al seu *email*. D'aquesta forma el signant rebrà al seu *email* la petició automàticament, entrarà amb el seu certificat digital i signarà sense haver-se registrat a l'aplicació. Un cop signat, l'emissor rebrà una còpia del document signat.

El document s'estructura en quatre capítols. En el primer capítol s'introduirà al lector en el context de la signatura electrònica, comentant a grans trets en que consisteix, que és un certificat digital i quins camps conté, que és una Autoritat de Certificació, i el protocol SSL (*Secure Socket Layer*).

En el primer, s'introduirà l'empresa Isigma i es descriurà el *software*, les seves llibreries de signatura digital i autenticació anomenades ISM (*Identity and Signature Managment)* i l'aplicació Portasigma a la que se li realitzarà la implementació de la nova arquitectura de signatura. També s'explicarà la problemàtica que ha aparegut pel fet de desactivar la tecnologia que permet

l'execució de Java a alguns navegadors i per que sembla que aquest problema és una tendència i no una decisió presa únicament per Google.

En el segon capítol, s'analitzaran un seguit de possibles tecnologies a implementar, es decidirà per una, i s'argumentarà el perquè de la decisió presa.

En l'últim capítol, s'explicarà la implementació de l'arquitectura escollida en el capítol 2. A més, es detallarà els nous procediments per autenticar i signar que s'han hagut de desenvolupar per assegurar que el sistema no és vulnerable.

CAPÍTOL 1. ISIGMA I EL SEU *SOFTWARE*

Isigma és una empresa amb més de 10 anys, especialitzada en la implantació de la signatura electrònica. En aquest primer capítol, es descriuran les diferents aplicacions PS (Portasigma) i CS (ClickSign) i la relació que tenen a través del conjunt de llibreries anomenades ISM *(Identity and Signature Management).* Per tal de poder comprendre el funcionament de les aplicacions ofertes per Isigma, però, s'introduiran en primer lloc els conceptes criptogràfics necessaris.

1.1. Conceptes criptogràfics

Per tal d'entendre el funcionament d'Isigma, cal conèixer alguns conceptes criptogràfics, com la signatura digital i les seves propietats, la CA (*Certification Autorithy)*, el protocol de seguretat SSL i la seva relació amb els certificats digitals.

1.1.1. Signatura digital

Una signatura digital és un procediment matemàtic que permet associar la identitat d'una persona a un document. També depenent del tipus de signatura, és capaç de validar l'autenticitat i la integritat del document. Generalment, aquesta es basa en criptografia de clau pública o criptografia asimètrica [1].

Hi ha diferents procediments per obtenir una signatura digital, però en general es basen en els següents passos:

- Creació de les dades a signar.
- Obtenció del digest. S'aplica la funció *hash* [2] a les dades a signar. Aquesta funció aplica un algorisme que resumeix les dades. És una funció unidireccional ja que a partir d'aquest 'resum', anomenat digest, no es pot recuperar el missatge original.
- Encriptació. S'encripta el digest amb la clau privada de l'emissor. Aquesta clau privada es genera junt amb la clau pública. La privada es manté en possessió de l'emissor i serveix per realitzar l'encriptació. La pública, que es transferida al receptor del missatge, serveix per desencriptar.

El resultat obtingut ja es pot considerar una signatura digital i serà enviat junt amb les dades originals i la clau pública de l'emissor.

Mitjançant la clau pública que l'emissor ha enviat, el receptor desencripta la signatura digital i obté el *digest*. Per verificar que el document rebut no ha estat alterat, es realitza la funció *hash* del document rebut i es comparen els dos *digests,* el rebut i l'obtingut. Si aquests són iguals, es pot afirmar que el document no ha estat modificat després de la signatura.

En la Fig. 1 es veuen els dos procediments, de generació i verificació de la signatura. Juntament amb el missatge i la signatura, l'emissor envia un certificat digital que conté les dades identificadores de l'emissor i la seva clau pública, necessària per verificar la signatura.

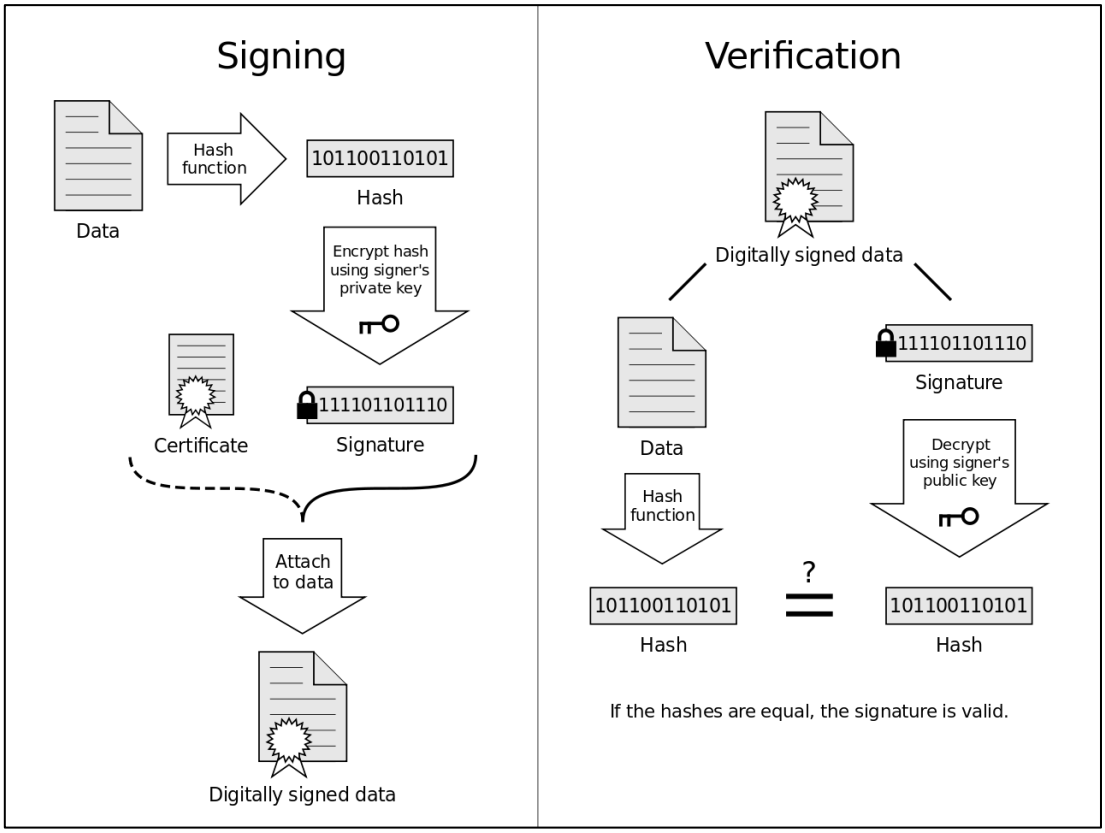

Fig. 1 Processos de signatura digital i verificació

1.1.2. Certificat digital

Un certificat digital és un document digital utilitzat per agrupar una clau publica amb un seguit de propietats i atributs específics del titular. Aquest titular pot prendre diferents formes com pot ser una persona, una organització, una web, o un *software*.

El certificat digital proporciona informació sobre l'emissor degut a que aquest certificat ve signat per una entitat superior que li dona credibilitat. Aquesta entitat s'anomena CA i ha d'estar reconeguda pel país o context on aporta aquest grau de credibilitat. Alguns dels atributs del certificat són:

- *Serial Number*: és l'identificador del certificat.
- *Signature algorithm*: és el nom de l'algorisme usat per signar.
- *Issuer*: nom de l'entitat que l'ha emès.
- *Valid from*: data d'inici de la validesa.
- *Valid to*: data de caducitat del certificat.
- *Subject*: nom de la persona o entitat que representa.
- *Public key*: és la clau publica associada al certificat.
- *Key usage*: usos de la clau pública, p.e. signatura o encriptació.
- *Extended key usage*: altres usos com p.e. representar una direcció web.

1.1.3. Autoritat de certificació

Una autoritat de certificació o CA, també anomenada prestador de serveis de certificació, és una entitat que actua com a tercers de confiança, encarregada d'emetre certificats junt amb la seva corresponent clau privada i pública.

La CA corrobora, abans d'emetre el certificat, que la persona o entitat que vol el certificat és realment qui diu ser. Un cop això, la CA emet el certificat signat amb la seva clau privada i s'afegeix a la cadena de certificació d'aquell certificat, amb lo qual, queda constància de qui ha estat l'emissor.

1.1.4. Cadena de certificació

Per tal de poder autenticar un usuari i obtenir-ne la seva clau pública, cal verificar tots els certificats de la cadena, cadascun dels quals ha estat emès per una CA de nivell superior. Al capdamunt de la cadena està el certificat arrel de CA. Aquest és un certificat auto signat i que assegura la veracitat de tota la cadena. Es tracta d'una jerarquia, a l'inici d'aquesta cadena està la CA arrel, i per sota altres CA però que depenen de l'arrel, i com últim esglaó es troba el certificat d'ús propi que no té potestat per signar altres certificats.

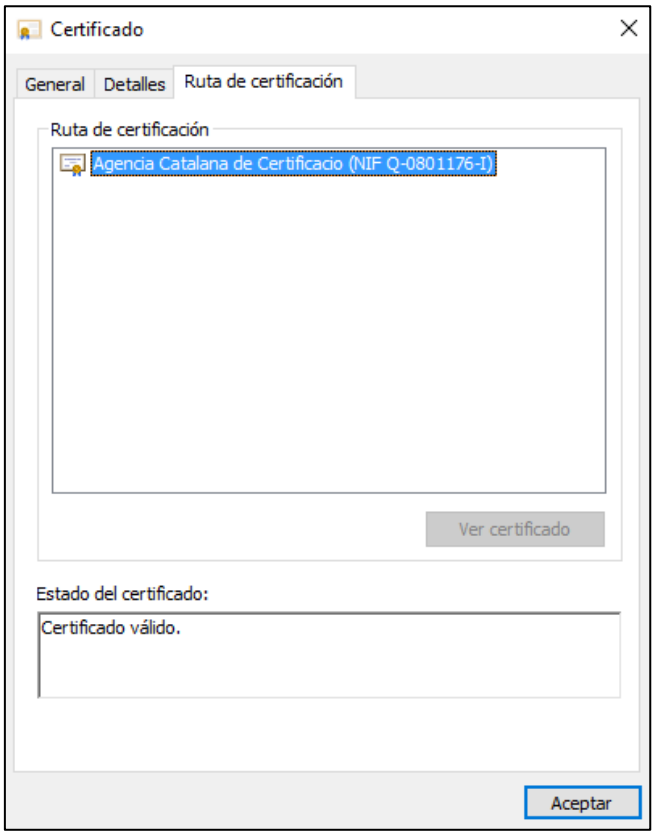

Fig. 2 Certificat de la CA arrel de l'Agencia Catalana de Certificació

A continuació es pot veure la cadena de certificació d'un certificat emès per una CA intermitja. El certificat de la CA intermitja ha estat emès per la CA de l'Agencia Catalana de Certificació.

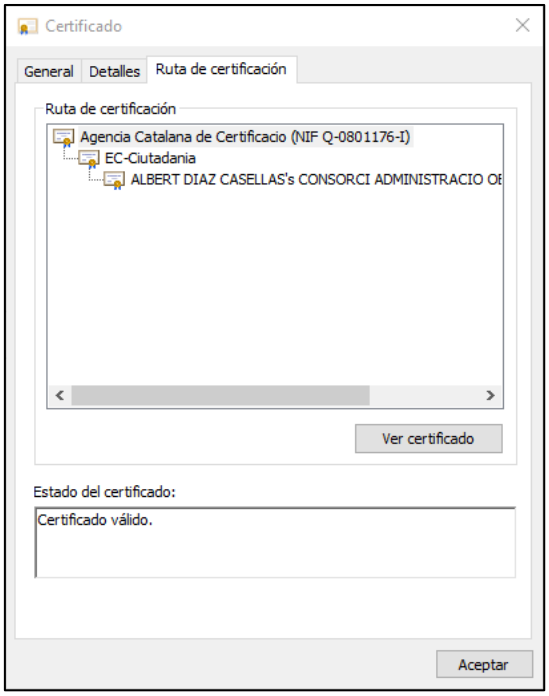

Fig. 3 Cadena de certificació d'un certificat emès per una CA intermèdia

1.1.5. PKCS#7

PKCS7 (*Public-Key Cryptography Standards 7*) [3] és un format per representar missatges protegits criptogràficament. La norma PKCS7 defineix unes estructures de dades per representar cada un dels camps que formen part del missatge. Aquest format accepta dos tipus de signatura, *attached i detached*. Una signatura és *attached* quan les dades del certificat i la signatura estan juntes i, en canvi, una signatura es considera *detached* quan s'inclou la signatura i el certificat per un costat i les dades signades per un altre.

1.1.6. Secure Socket Layer

Secure Socket Layer (SSL) és un protocol criptogràfic que proporciona comunicacions segures. Utilitza certificats i per tant, criptografia asimètrica (clau pública – clau privada) per autenticar-se. Aquest protocol té tres fases bàsiques:

- Negociació entre les parts de l'algoritme que s'utilitzarà. p.e: RSA [4].
- Intercanvi de claus públiques i autenticació basada en certificats.
- Xifrat del tràfic basat en xifrat asimètric.

Aquest protocol, va evolucionar cap al TLS (*Transport Security Layer*) [5], i les primeres versions de SSL 1.0., 2.0 i 3.0 totes ja es consideren insegures i no s'utilitzen.

1.2. Presentació d'Isigma

Isigma té diferents aplicacions segons la necessitat del client, com PS per a un procés de signatura *cloud* o CS per a realitzar signatures des de l'escriptori, les

dues basades en les llibreries pròpies ISM. També té desenvolupat un client per als *webservices* de PS que permet pujar documents amb les peticions de signatura corresponents, anomenat PS-Dispatcher. En la Fig. 4 es pot observar un esquema de l'arquitectura de l'empresa amb el software abans mencionat. En els següents apartats s'entrarà més en detall en les dues aplicacions principals implicades en aquest treball, PS i CS.

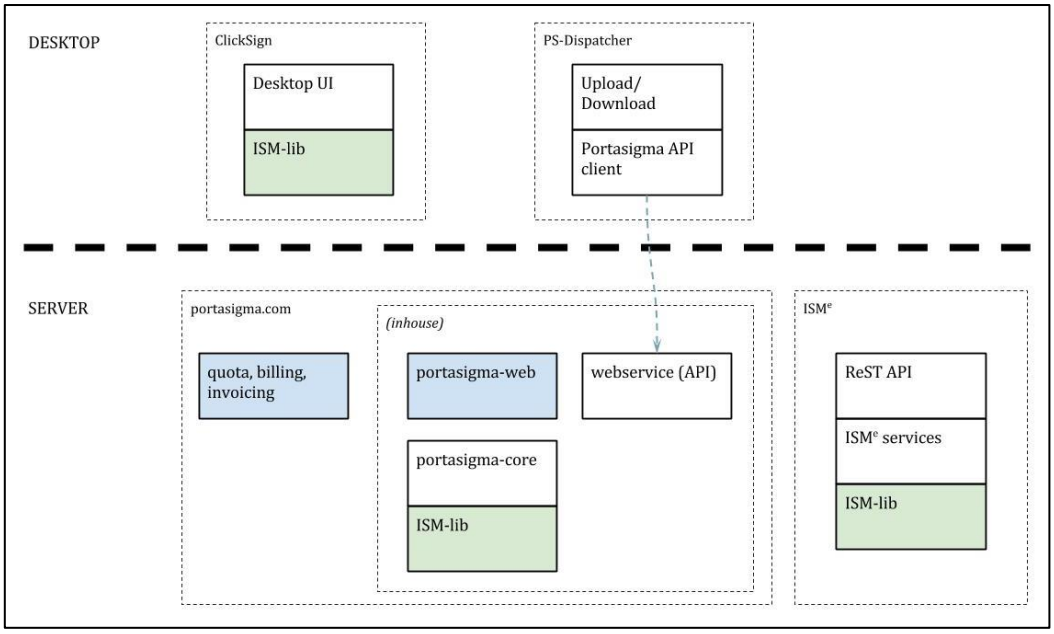

Fig. 4 Esquema de l'arquitectura de la tecnologia d'Isigma

1.3. Descripció de ISM

ISM o *Identity and Signature Management* és el conjunt de llibreries que contenen la lògica per a realitzar signatures digitals. Es tracta d'un software modular dissenyat per a que altres aplicacions l'integrin i puguin aprofitar el seu potencial per a realitzar signatures digitals.

1.3.1. Arquitectura

L'ISM es divideix en diferents mòduls, cadascun diferenciat per la seva funció:

- *Core*: Té les interfícies bàsiques, els suports per als dispositius, la configuració, el tractament d'excepcions...
- *Sig*: És el mòdul encarregat del tema de la signatura en si, realitza signatura tant en fitxers PDF (Portable Document File), XML i plans, en formats PAdES [6], XAdES [7], CAdES [8], PKCS7...Verifica la signatura, valida els certificats, genera la signatura i la validació amb *timestamp*...
- *Auth*: S'encarrega de la lògica per a l'autenticació amb certificats digitals. És l'únic punt d'entrada a la lògica d'autenticació, gestiona si escollir els certificats des del sistema operatiu o si també hi ha connectat al PC un dispositiu lector de targetes criptogràfiques. Acaba depenent dels mòduls crypt i sig.
- *Crypt*: Conté les classes necessàries per gestionar tot el tema de l'encriptació: simètrica, asimètrica, *mixed*...i de generar les excepcions

corresponents a possibles errors que es puguin generar amb els tipus d'encriptació.

- *License:* Aquest mòdul conté les classes encarregades de portar el control de les llicències.
- *Sig-utils*: Conté utilitats complementàries al mòdul sig.
- *License-utils:* Conté utilitats complementàries al mòdul *license*.
- *Win-menu*: Es tracta d'un conjunt de classes per a tenir una interfície visual en sistemes operatius Windows.
- *Gui:* Conté els formularis que utilitzarà el mòdul *win-menu* per a la interfície visual de Windows. Si hi hagués una versió per un altre sistema operatiu es crearia un altre mòdul que utilitzaria el mòdul *gui.*
- *Autosign*: Aquest mòdul permet realitzar un procés de signatura automatitzada de tots els fitxers que es deixin en una carpeta.

En la Fig. 5 es pot veure un esquema dels diferents mòduls abans mencionats.

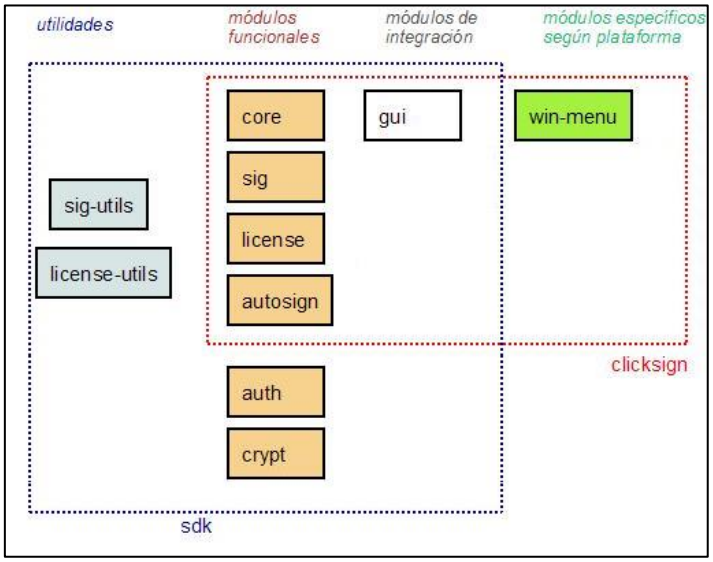

Fig. 5 Esquema modular de l'ISM

1.3.2. ClickSign

CS és una aplicació d'escriptori per a usuari final, que principalment permet realitzar signatura de fitxers PDF, XML i plans. Està formada per un conjunt de certs mòduls de l'ISM com es pot veure a la Fig. 5.

1.3.2.1. Funcionalitats

L'aplicació té diverses funcionalitats, a més de realitzar signatures, permet verificar-les a través de l'opció "Verificar" del menú contextual.

- L'opció "Firmar" és l'encarregada de signar el document.
- L'opció "Verificar" permet verificar que una signatura sigui correcta, i que el document no s'ha modificat un cop ha estat signat.
- A l'opció "Configuració" s'obra la finestra de configuració Fig. 6 de l'aplicació, amb varies pestanyes on s'introdueix la llicència, el mode de

signatura PKCS7 en *attached* o *dettached*, les opcions de la firma visible, diagnòstic d'errors, configuració de SignLoop, etc.

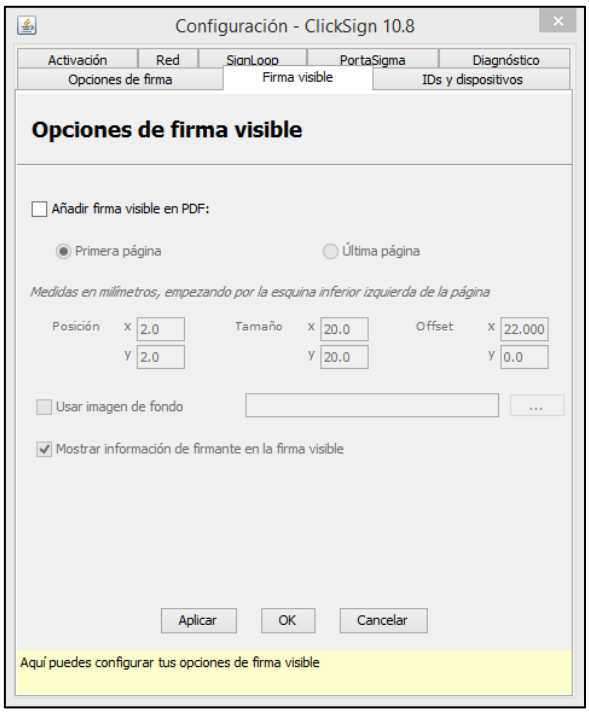

Fig. 6 Finestra de configuració de CS

- Si es clica l'opció "AutoSign", s'executa un petit programa en *background*, que va realitzant un escaneig cada cert interval buscant documents que puguin ser signats, si ho aconsegueix els signa amb el certificat que s'hagi configurat i els desa a la carpeta de sortida.
- L'opció de "Solicitar Firmas" permet sincronitzar el CS amb PS de forma que pot pujar documents a través de la seva API [9].

1.3.2.2. Descripció d'ús

CS té dues formes de funcionar, amb interfície gràfica o a través d'una finestra de comandes.

Interfície gràfica

Per utilitzar aquest mètode només cal clicar amb el botó dret del ratolí a sobre del document que es desitja signar. Apareixerà el menú contextual de Windows, però modificat amb una entrada anomenada "ClickSign", i situant-se a sobre es

desplegarà el menú amb les diferents opcions de l'aplicació com es veu a la Fig. 7.

Fig. 7 Menú contextual de CS a Windows

Si es clica l'opció Firmar, s'obra una finestra per elegir el certificat (Fig. 8) entre els instal·lats al sistema operatiu o en algun dispositiu criptogràfic connectat al ordinador.

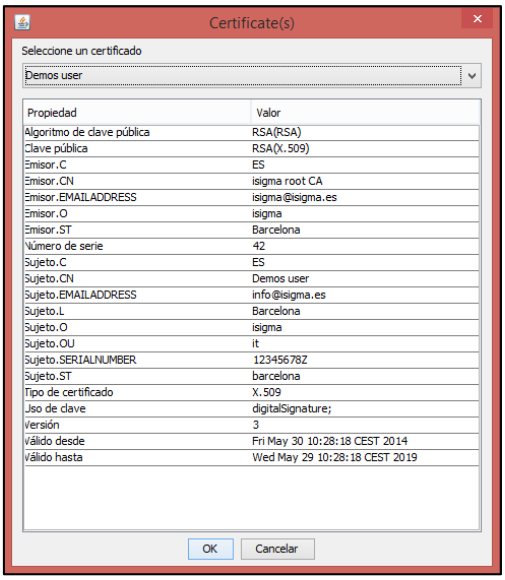

Fig. 8 Finestra de selecció de certificats

Es selecciona es clica OK, i s'obté una copia del fitxer original amb la signatura i amb extensió *.signed.* (Fig. 9)

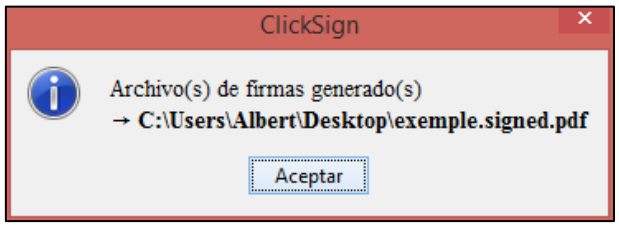

Fig. 9 Feedback del procés de signatura

Ara es pot utilitzar la funcionalitat de verificar la signatura per a comprovar la veracitat d'aquesta. Per fer-ho es clica botó dret sobre el document signat, s'elegeix l'opció ClickSign/Verificar i s'obrirà una finestra amb el resultat de la verificació, com es veu a la Fig. 10.

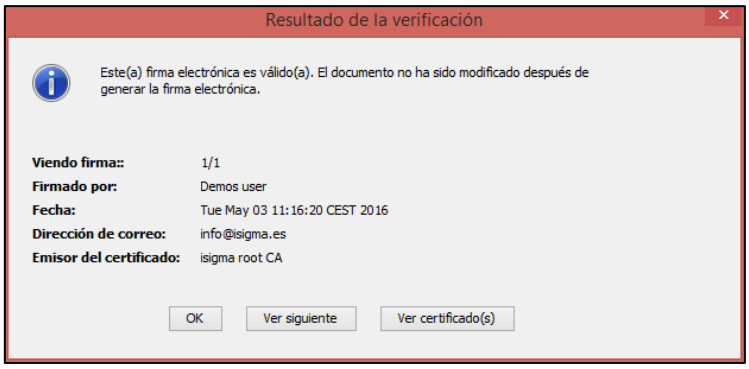

Fig. 10 Resultat de la verificació de la signatura

Consola

Per a un ús més automatitzat l'aplicació també permet totes les funcionalitats mostrades al menú contextual des d'una consola. A la Fig. 11 es mostra una crida d'exemple per a realitzar una signatura amb un certificat instal·lat al repositori del sistema operatiu que com a DNI té 12345678Z.

```
ismrun es.isigma.ism.winmenu.SignFiles --certificate
        ".*SERIALNUMBER=12345678Z.*" --pin ismdemo --out-file
        fitxerSignat.pdf -no-gui fitxerOriginal.pdf
```

Fig. 11 Comanda d'exemple per signar un fitxer amb CS

1.3.2.3. Instal·lador amb tecnologia NSIS

CS utilitza la tecnologia NSIS (*Nullsoft Scriptable Install System)* per definir el seu instal·lador. Gracies a NSIS pot definir el numero de pàgines que tindrà la instal·lació, l'idioma de l'aplicació...Però, a més d'aquestes funcionalitats, NSIS el que permet es executar comandes a la consola de Windows per així realitzar la distribució dels fitxers a les carpetes corresponents, o canviar configuracions del sistema operatiu. Per exemple, amb la nova instal·lació que s'ha realitzat en aquest projecte s'autoinstal·la un certificat al repositori de Windows i es modifica el registre del sistema operatiu.

1.3.3. Compilació

Per compilar tant el SDK de l'ISM que té integrat PS com el *software* CS (Fig. 12), s'utilitza l'eina Apache ANT [10]. Per realitzar-ho, hi ha preparats uns *scripts* per cada cas, on solament se'ls ha de passar com argument la versió.

```
@echo off
if "%1" == "" goto usage
call ant -Dmanufacturer=isigma -Dvariant=clicksign -
Dversion=%1 clean prepareOEM clicksign
goto end
:usage
echo [ERROR] Usage: %0 version
:end
```

Fig. 12 Script per realitzar la compilació de CS

1.4. Descripció de Portasigma

PS és una aplicació web de signatura electrònica desenvolupada per Isigma. La seva principal funció és la realització de signatura de documents PDF en el *cloud*.

1.4.1. Descripció d'ús

En aquest apartat es descriurà PS des del punt de vista d'usuari. Les modificacions que en capítols posteriors es presentaran són a nivell tècnic, i per tant l'usuari no veurà cap diferència tot i que la tecnologia utilitzada hagi estat substituïda.

Hi ha dues formes d'autenticació a PS: amb usuari i contrassenya i amb un certificat digital reconegut. L'autenticació amb usuari i contrassenya és per a usuaris registrats en l'aplicació. Entrant per aquest punt a l'aplicació, l'usuari pot crear peticions de signatura i signar amb un certificat que prèviament haurà pujat i associat al seu compte d'usuari en format PKCS12 [11].

En la Fig. 13 s'observa la pàgina d'autenticació amb usuari i contrassenya:

Fig. 13 Pàgina d'inici de PS

D'altra banda, amb l'autenticació amb certificat digital reconegut, només es poden signar documents. Per fer ús del certificat digital, la pàgina executa un *applet* que accedeix al repositori del sistema operatiu on es troba instal·lat el certificat o al dispositiu criptogràfic, si aquest s'està utilitzant. Un cop l'*applet* ha obtingut el certificat, el valida mitjançant OCSP (*Online Certificate Status Protocol*) [12] i finalment els mostra a la pàgina web (Fig. 14).

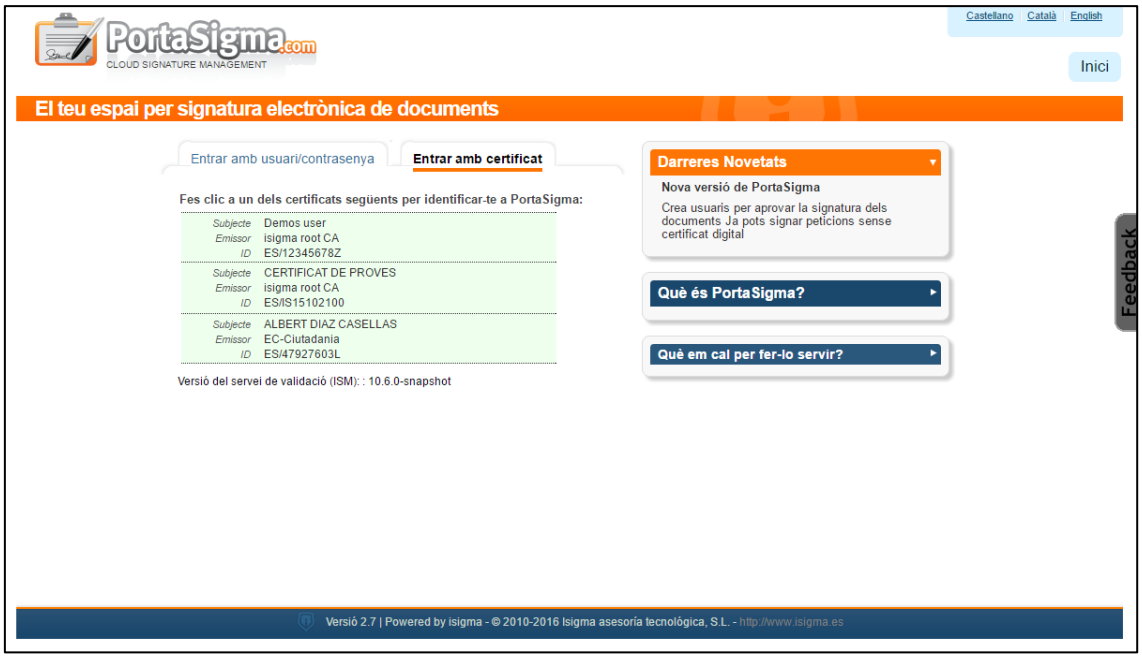

Fig. 14 Pàgina amb els certificats obtinguts per l'*applet*

Segons el tipus d'autenticació realitzada s'accedeix a un menú o a un altre. Si s'ha entrat amb usuari i contrassenya s'accedeix a aquesta pàgina amb les tres opcions que es mostren a la Fig. 15, i si s'ha entrat amb certificat només es disposarà de l'opció de "Les meves signatures", tal i com es veu a la Fig. 16.

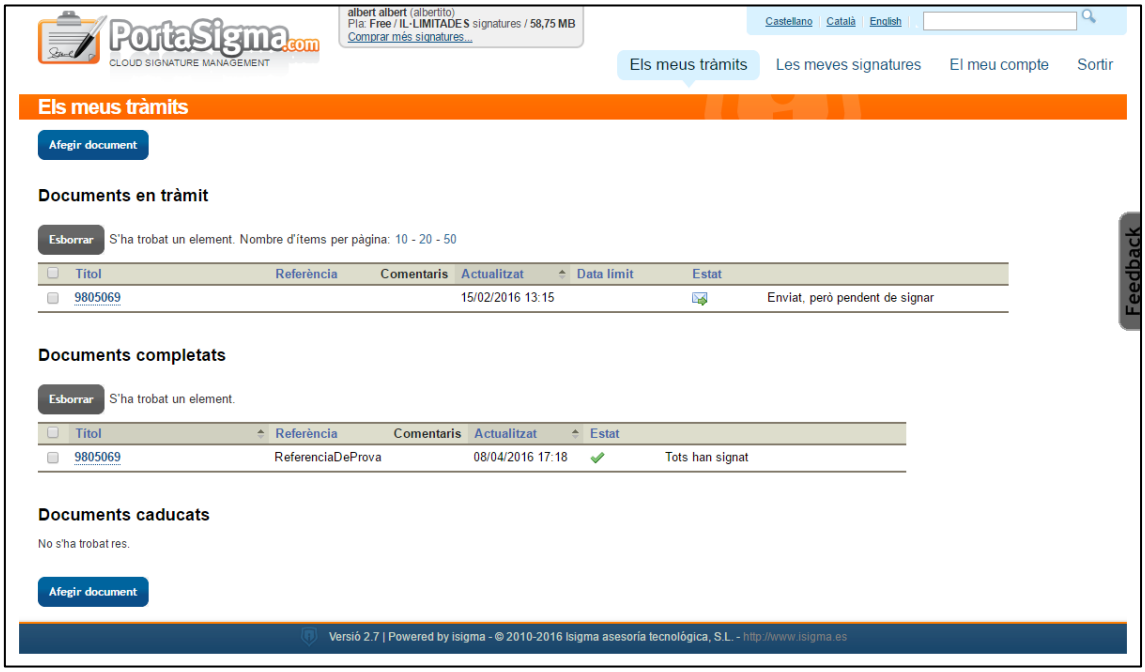

Fig. 15 Opcions amb l'autenticació amb usuari i contrassenya

Un cop rebuda la petició de signatura assignada al nostre ID, que en el cas d'Espanya seria el DNI, s'usa la tecnologia de l'*applet* per signar, i s'entra amb el certificat. En aquest punt hi ha dues formes de realitzar la signatura, simple o múltiple, depenent del nombre de documents que es desitgi signar.

Signatura simple

Es selecciona la petició de signatura per entrar a signar com es veu a la Fig. 16.

Fig. 16 Opcions amb l'autenticació amb certificat

Un cop a la pàgina de signatura, es revisa que el document sigui correcte, s'accepten les condicions i es clica el botó "Signar".

Fig. 17 Pàgina de signatura simple a PS

Signatura múltiple

Si es desitja realitzar un conjunt de documents, es pot escollir l'opció de signatura múltiple, en la Fig. 16 es veu com seleccionar aquesta opció a través d'un enllaç on diu "Signatura Múltiple". Si es clica aquest enllaç, permet seleccionar múltiples

documents a signar, tal i com es veu a la Fig. 18, on hi ha un document de color rosat que s'està signant i tres en groc indicant que estan bloquejats perquè en breu seran signats. D'altra banda, els documents que apareixen en color verd són els documents ja acabats de signar (Fig. 19).

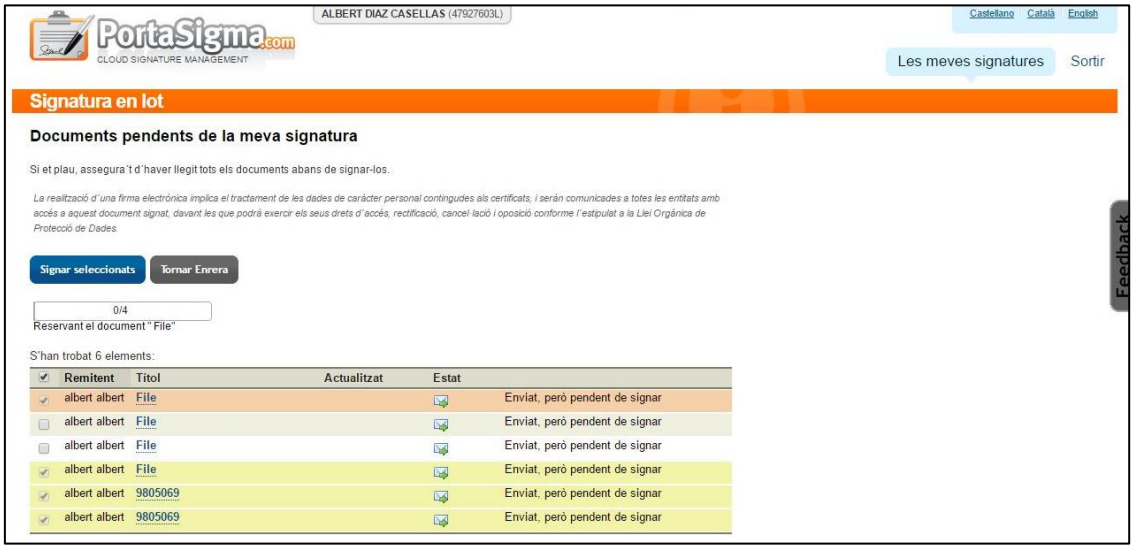

Fig. 18 En procés de la signatura múltiple

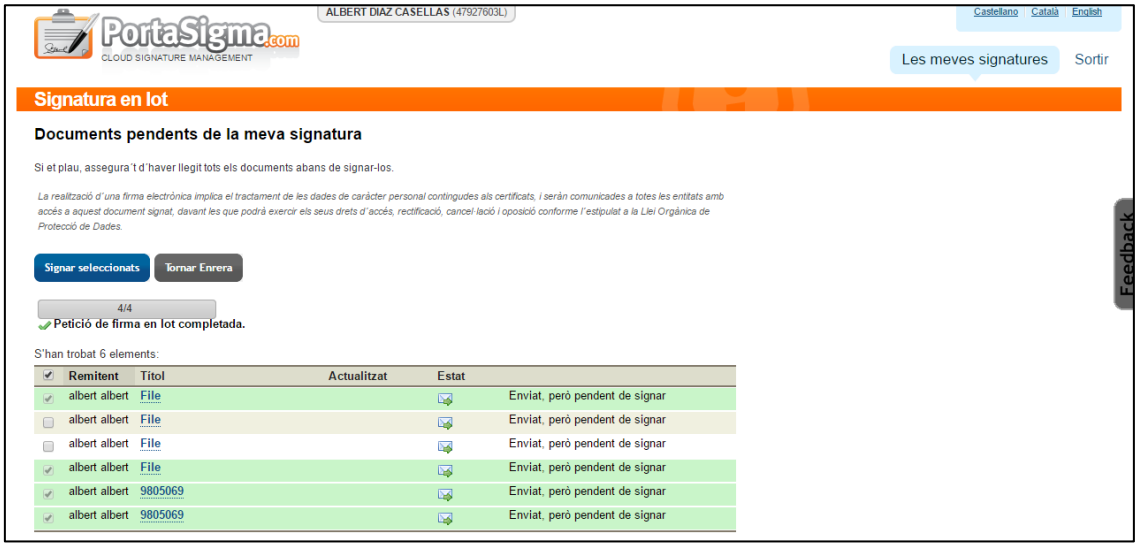

Fig. 19 Procés de signatura múltiple realitzat

1.4.2. Descripció tècnica

PS és una aplicació web, i com a tal es desplega en un servidor web, en aquest cas es fa en un Tomcat (veure secció 12 de l'Annex). L'aplicació té dues versions, *inhouse* per a realitzar-li la instal·lació a un client que el compri i *quota*, la qual és la versió al servidor de l'empresa en la qual, per a que un usuari pugui crear sol·licituds de signatura han de pagar una quota per a cada signatura, venuda en paquets de signatures.

1.4.2.1. Tecnologies

A continuació es detallen les principals tecnologies utilitzades a PS:

- Appfuse. És el *framework* central, l'esquelet en el qual es basa. Conforma el conjunt d'eines necessàries per a crear qualsevol tipus d'aplicació web *[13]*.
- Struts2. *Framework* complementari a Appfuse. Dota a PS dels serveis necessaris per un us empresarial *[14]*.
- Spring. D'aquest *framework* s'utilitza la part de seguretat, la qual s'encarrega de gestionar els accessos als contexts de l'aplicació en funció dels rols de cada usuari *[15]*.
- Hibernate. *Framework* utilitzat com a gestor de connexions a la base de dades *[16]*.
- Maven. És l'eina encarregada de realitzar la compilació de l'aplicació, opcionalment executar els tests, generar els fitxers amb extensió jar i el desplegament a un servidor remot *[17]*.

1.4.2.2. Arquitectura

PS està format per 7 projectes, els quals tenen les seves funcionalitats ben diferenciades.

- Portasigma-core: Projecte on estan definides les classes del model, la capa de servei, les notificacions ...
- Portasigma-web: Projecte on estan incluides les classes relacionades amb la part web, estan les webs, els scripts, els fitxers CSS (Cascading Style Sheets) [18], les classes *interceptors*, els accessos a les URLs, els contenidors d'objectes de Spring...

Els projectes amb '*invoicing'* o el sufix 'quota' només estan afegits al desplegament *quota* ja que les seves funcionalitats estan orientades per al ús del pagament per signatura.

- Invoicing-core: Projecte on estan definides les classes pertanyents al tema de la facturació dels usuaris.
- Invoicing-web: Part web del tema de la facturació, per a gestionar l'aplicació amb l'usuari administrador.
- Portasigmaquota-core: Inclou els plans de preus, capacitat d'emmagatzematge de documents per usuari, gestió d'associacions...
- Portasigmaquota-web: Conté la part dels usuaris d'associacions i les webs corresponents. És el projecte principal a desplegar al servidor, i té totes les dependències afegides de la resta de projectes, per d'aquesta forma poder afegir a totes les funcionalitats de l'aplicació.
- Portasigma-ws: En aquest projecte està definida la API REST que serveix per pujar documents, crear peticions de signatura i per realitzar una descarrega dels documents signats a traves d'uns *webservices*.

A la Fig. 20 està definida l'arquitectura de PS. Es veu com la part *inhouse* engloba solament els projectes portasigma-web, portasigma-core i portasigma-

ws, a més de les llibreries relacionades amb la criptografia anomenades ISM. Mentre que l'aplicació quota, si que conté tot, els projectes quota, *billing* i *invoicing*, a més dels de l'*inhouse*.

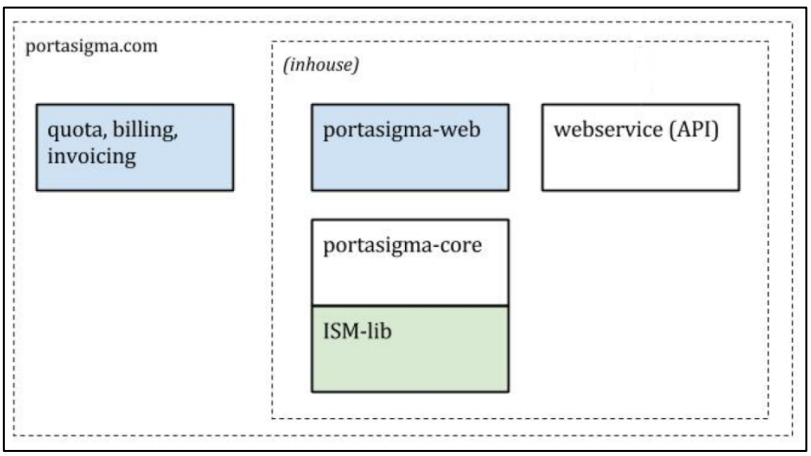

Fig. 20 Arquitectura de PS

1.5. Problemàtica de la solució actual

Abans d'explicar la problemàtica cal entrar en context i per això s'ha d'explicar la tecnologia NPAPI *(Netscape Plugin Application Programming Interface)* [19].

NPAPI és una API *(Application Programming Interface)* que permet desenvolupar extensions per navegadors. En l'arquitectura NPAPI, un *plugin* qualsevol declara un *content type*. Quan el navegador es troba amb un *content type* que no sap obrir, carrega el *plugin* adequat i l'executa des de dins de la pròpia pàgina. Llavors el que actua és el *plugin* realitzant la lògica que correspongui i entregant les dades al navegador.

Degut a l'antiguitat i els problemes de seguretat que té aquesta tecnologia, a més de la consolidació de la web sense *plugins* amb HTML5 *(HyperText Markup Language 5)*, els principals navegadors van introduir en els seus fulls de ruta la desactivació d'aquesta tecnologia.

El primer a donar el pas va ser Google Chrome amb la versió 45, publicada el mes de Setembre de 2015. Encara que per defecte ja venia desactivat des de la versió 42, els usuaris encara podien activar NPAPI des del menú de configuració del navegador.

Posteriorment, es van sumar diferents navegadors a l'opció presa per Google, tals com Mozilla Firefox, el qual planeja fer-ho a finals de 2016, y Opera, on ja no l'accepta des de la versió 33.

Aquest fet, com s'ha dit abans, és una tendència ja que Microsoft en el seu nou navegador Edge, des de la seva primera versió ja no introdueix aquesta tecnologia.

Aquesta decisió, es presa degut a la gran quantitat de problemes de seguretat que té aquesta tecnologia.

Pel que fa a la nostra aplicació PS, aquesta tecnologia permetia executar els *applets* Java per a poder accedir als certificats de l'usuari, instal·lats al sistema operatiu o en un dispositiu criptogràfic, des del navegador. Arrel de treure aquesta tecnologia, l'aplicació no té cap mètode per accedir als certificats de l'usuari i per tant queda impossibilitada la signatura amb certificat des del ordinador de l'usuari.

CAPÍTOL 2. ARQUITECTURES POSSIBLES

Actualment hi ha diferents alternatives per substituir els *applets* però cap d'elles predomina sobre les altres:

- Complements per navegador Web
- Invocació per protocol
- Comunicació per servei
- Invocació per protocol i comunicació per servei
- API Javascript

A continuació es detallaran les diferents tecnologies, destacant els avantatges i els inconvenients des dels diferents punts de vista dels implicats, desenvolupador, integrador i de l'usuari.

2.1. Complements per navegador Web

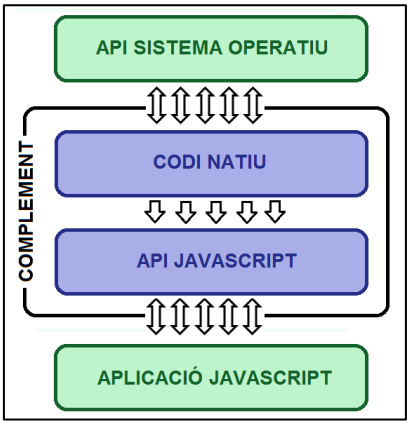

Fig. 21 Integració del complement web

La API del sistema operatiu és l'encarregada de proporcionar l'accés al repositori de claus i certificats de l'usuari. Per tant, el codi natiu haurà d'estar perfectament adaptat a la API per una banda, i per l'altra (Fig. 21), a la API Javascript (JS) per poder tenir una perfecta interacció amb tots els navegadors i alhora amb l'aplicació web.

En aquest cas, l'arquitectura resultant seria bastant diferent a una aplicació basada en un *applet* ja que mentre en una quasi tota la firma es realitza seguint la API JSE (*Java Standard Edition*), l'altra es basa en codi JS [20] que s'executa al navegador.

Essent la part menys portable de les aplicacions i la més complexa d'actualitzar, seria desitjable desenvolupar el mínim codi pel que fa al complement i que no depengués de normatives.

Un exemple de funcionalitats bàsiques que hauria de tenir podria ser el següent:

- Signatura PKCS1, codificada en base 64 [21].
- Obtenció del certificat, codificat en base 64.
- Obtenció de referència a la clau privada.

Aquests funcionalitats estan més detallades en un estàndard de W3C (*World Wide Web Consortium*) anomenat WebCrypto Key Discovery [22], per tant, s'hauria d'implementar part d'aquesta API complementada amb codi JS.

2.1.1. Aspectes de seguretat

Tractament de sessions

S'han d'implementar mecanismes que assegurin que les referències obtingudes a les claus privades siguin d'un únic ús limitat a la sessió i al context JS des d'on es van demanar, encara que els diferents contexts provinguin del mateix domini web.

Confirmacions per part de l'usuari

L'usuari ha d'ésser plenament conscient de quan l'aplicació web vol accedir als seus certificats o a les seves claus privades, i també quan realitza operacions amb aquelles claus. Per tal d'aconseguir això, s'han de realitzar uns *pop-up's* (Fig. 22) sol·licitant permisos.

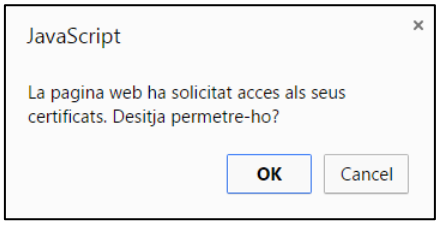

Fig. 22 Avís en el navegador

Concepte de sessió i context JS

S'entén per context Javascript l'objecte en que està contingut el codi en execució. Usualment, aquest context és l'objecte *Window* de JS, però si les referències i certificats de claus es sol·liciten des d'un context inferior, com pot ser dins d'una funció, no es pot arribar a accedir a aquesta referència des d'un context superior, com seria el global en aquest cas, però si des d'inferiors.
És a dir, s'ha de tenir un control absolut de les referències i certificats de claus, i no permetre l'accés a aquests paràmetres des de objectes més generals per tal que codi malintencionat hi tingui accés.

2.1.2. Problemàtica en la homogeneïtat dels complements segons navegador web

Actualment, els navegadors implementen diferents tecnologies de complements, això obliga a implementar un possible complement per cada una d'elles.

Anàlisi segons la tecnologia del complement

Pel que fa als tres principals navegadors, Google Chrome utilitza el seu propi estàndard *PPAPI (PepperAPI)*, Mozilla Firefox usa *NPAPI*, i Internet Explorer ha eliminat l'ús de complements.

Variacions de navegador Web per sistema operatiu

En un mateix navegador el suport de complements tampoc no és uniforme segons el sistema operatiu: hi ha versions antigues que no suporten alguns complements. En la Taula 2.1 es mostra el suport de complements en funció dels sistemes operatius i navegadors.

Taula 2.1 Relació entre navegador web i sistema operatiu

Google Chrome	Microsoft Windows	OK
	M.Windows Modern UI	NO
	Apple OS X	OK
	Linux	OK
Apple Safari	Microsoft Windows	OK
	Apple OS X	OK
Mozilla Firefox	Microsoft Windows	OK
	Apple OS X	OK
	Linux	OK

Canvis per sistema operatiu

Una altra característica a tenir en compte al desenvolupar el codi, és la API que té cada sistema operatiu per accedir a les claus criptogràfiques. Cadascuna té els seus mètodes corresponents anomenats en la Taula 2.2.

Taula 2.2 Sistema operatiu i nom del seu corresponent repositori local

Sistema operatiu	Nom del repositori
Apple OS X	Clauer de OS X
Microsoft Windows	Microsoft CryptoAPI
Linux	NSS (Name Service Switch)

Accés a targetes intel·ligents

Pel que fa a l'accés a les targetes intel·ligents, cadascuna d'aquestes APIs d'accés a les claus criptogràfiques pot necessitar desenvolupaments addicionals, ja que cadascuna poden entregar la informació del certificat en diferents formats.

Taula 2.3 Accés a targetes intel·ligents segons la API del sistema operatiu

Nom del repositori	Classe
Clauer d'Apple OS X 10.9 i anteriors	Tokend
Clauer d'Apple OS X 10.10 i superiors	CryptoTokenKit
Microsoft CryptoAPI	API Integrado
NSS	PKCS#11

Tal i com es mostra a la Taula 2.3, segons la versió del sistema operatiu d'Apple la classe utilitzada per accedir al repositori de claus es *Tokend* o *CryptoTokenKit*.

Sol·licitud del PIN o contrassenya dels repositoris

S'ha de realitzar una gestió del PIN o de la contrassenya dels repositoris de claus de forma que ho gestioni el sistema operatiu o una petició externa. En cas de petició externa, es necessitaria crear el codi necessari per tenir una interfície gràfica adequada al sistema operatiu, que a més fos accessible per persones amb discapacitats que no poden memoritzar el PIN o contrassenya. En la Taula 2.4 es mostra en quins casos seria necessari introduir la petició externa o si pel contrari el sistema operatiu ja ho té integrat.

Taula 2.4 Integració de la gestió del PIN dels repositoris

Repositori del sistema	CryptoAPI	Integrat
	Clauer de OS X	Integrat
	Repositori JCA/JCE	No integrat
	NSS	No integrat
Targeta intel·ligent	Tokend	Integrat
	CyptoTokenKit	Integrat
	PKCS#11	No integrat
	Via API: NFC (*Near Field Communication)* [23], PC/SC (*Personal Computer / Smart Card)* [24]	No integrat

2.1.3. Conclusions

L'ús de complements no és una bona opció, ja que el desenvolupament d'un complement per a navegadors depèn del sistema operatiu i del navegador a escollir.

D'altra banda, hi ha un punt més que desaconsella l'ús de complements: la seguretat. Tradicionalment, els complements han estat una font de problemes de seguretat, i fins i tot complements importants com Oracle Java han arribat a ser considerats com insegurs. Un complement que obre la porta al codi natiu privilegiat des del Javascript accessible des de pagines web, és una enorme responsabilitat en temes de seguretat. Això comporta actualitzacions de codi constants, i requereix donar facilitats per fer actualitzacions tan bon punt com es tingui desenvolupada una versió nova.

2.2. Invocació per protocol

Els sistemes operatius actuals tenen una sèrie d'associacions entre el tipus de fitxer i les aplicacions que són capaces d'utilitzar-los. Per exemple, en Windows, al obrir un arxiu amb extensió *txt*, aquest consultarà el Registre de Windows per saber quina és l'aplicació per defecte associada a aquesta extensió, i obrirà l'aplicació passant-li com a paràmetre la ruta completa del fitxer en l'esquema d'arguments definit en el Registre de Windows.

En canvi, per Linux, no es consulta l'extensió, es consulta el MIME-Type (*text/plain* en aquest mateix exemple).

Aquest mateix esquema es defineix d'igual manera en la majoria de sistemes operatius per esquemes comuns de protocols basats en URN/URI/URL (*Uniform Resource Name*/Identifier/Locator). Per exemple, si en Windows executem "*start http://www.google.es*" en la consola, s'iniciarà el navegador per defecte, que és l'aplicació associada per tractar el protocol *HTTP* (*HyperText Transport Protocol*), i es carregarà la pàgina web.

Aquesta forma d'obrir aplicacions es coneix com invocació per protocol o *"protocol handler"* [25], i de forma anàloga a la invocació per obrir aplicacions indicant un fitxer, on abans es rebia una ruta completa del fitxer a obrir, ara es rep la URN/URL/URI completa que es va indicar obrir.

2.2.1. Invocació en navegador web

Aquest sistema d'invocació per protocol des dels sistemes operatius és accessible des del navegador Web. Per tant, si en la barra de direccions del navegador s'indica una URI, el navegador traslladarà el control al sistema operatiu de forma que aquest localitzi l'aplicació apropiada per tractar el protocol associat a aquesta URI, i la obri passant-li aquella URI.

Un clar exemple d'això és Apple iOS. Si en el navegador s'escriu la URN *"tel:///1-408-555-5555"* on el número és un telèfon, una crida a una pàgina web amb una sentencia HTML així: *"1-408-555-5555",* provoca que s'activi el telèfon i es realitzi una trucada a aquell número, ja que l'aplicació nativa de telèfon de iOS té registrat aquest esquema de protocol.

En els casos, en que el navegador sàpiga tractar el protocol, com per exemple *HTTP, HTTPS (HyperText Transfer Protocol Secure), FTP (File Transfer Protocol)...,* no li transferirà el control al sistema operatiu.

Advertències d'apertura

En general, tots els navegadors Web mostren algun tipus d'advertència al obrir un contingut. Aquesta invocació per protocol no deixa de ser una transferència de dades des d'una pàgina Web a una aplicació nativa, per tant, els navegadors solen advertir a l'usuari com es mostra en la Fig. 23.

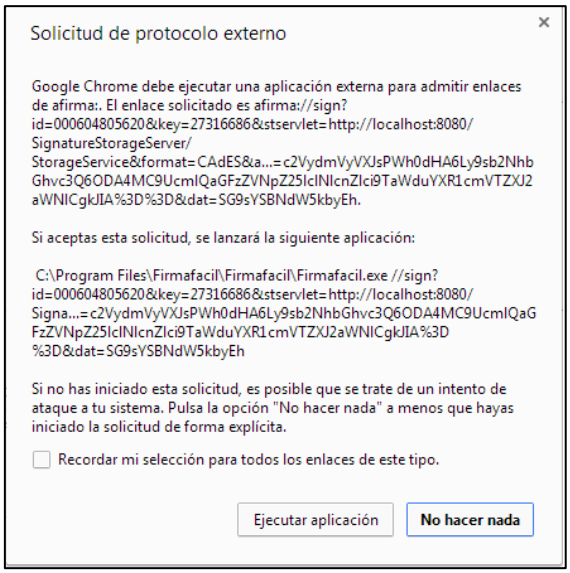

Fig. 23 Advertència d'apertura a través d'un *protocol handler*

En el cas d'Internet Explorer, a més, sempre que s'executa un programa des d'un *protocol handler* es comprova la signatura electrònica de l'executable del programa natiu invocat. No obstant, les signatures han d'ésser sempre executables en format PE *(Portable Execute)* de Microsoft. Per tant, si la aplicació invocada està desenvolupada en Java, com podria ser el nostre cas, realment l'aplicació que s'invoca no serà la pròpia nativa (un JAR de Java), sinó el propi JRE (*Java Runtime Environment*), i la seva signatura electrònica no pot ser controlada, ja que, el JRE l'instal·la un proveïdor aliè com Oracle.

Per tal de superar aquest requeriment que exigeix Internet Explorer, existeix l'aplicació Launch4J [26], que és capaç de crear executables del tipus PE a partir d'un JAR de Java. Un cop creat, s'hauria de signar amb un programa específic com Sign4J ja que el signador PE de Microsoft (SignTool) no l'accepta.

Suport a la invocació per protocol en els navegadors

Tots els navegadors accepten els *protocol handlers* excepte Google Chrome en versió Windows 8 UI (*User Interface*) *Modern*.

2.2.2. Invocació per protocol com a substitut dels *applets*

En utilitzar la invocació d'una aplicació externa al navegador s'han de considerar uns punts de cara a l'usuari:

- Una invocació d'una aplicació des del navegador Web provoca un canvi de context gràfic, que pot molestar a l'usuari. A més, al tancar-se l'aplicació s'ha de tornar al flux lògic de la pàgina web.
- No és possible tenir un UI integrat en la pàgina web, com succeeix amb els *applets*.
- Un cop s'invoca l'aplicació des del navegador, no pot existir una comunicació bidireccional directa entre els dos, com succeeix amb els

applets. No obstant, l'aplicació nativa pot rebre informació en la pròpia URL d'invocació.

Comunicació entre l'aplicació nativa i l'aplicació Javascript

Una aplicació JS pot invocar una aplicació nativa sempre que estigui registrada com aplicació per defecte per tractar un protocol que el propi navegador no sàpiga tractar, proporcionant les dades necessàries per l'aplicació nativa com part de la propi URI d'invocació, però ara falta establir el canal de tornada. Un exemple seria, registrar un protocol d'invocació amb extensió sign-cert, de forma que quan el navegador des de l'aplicació JS tingués que obrir la URI "sign-cert:certificat-digest", aquest al no poder-la interpretar delegaria l'extensió "sign-cert" cap al sistema operatiu i executaria l'aplicació nativa que rebria el certificat i el digest passats en la pròpia URI al fer la crida des del navegador.

Una forma senzilla és utilitzar un servidor intermediari que sigui accessible per les dues parts, tal com es mostra en la Fig. 24.

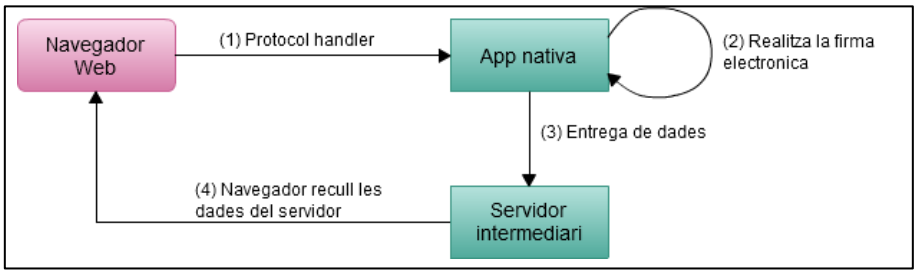

Fig. 24 Esquema de la comunicació entre l'aplicació i el JS

No obstant, per implementar aquest model és necessari aplicar certes mesures de seguretat:

- El servidor intermediari i l'aplicació Web han d'estar preferentment allotjats en el mateix servidor per evitar advertències de *cross site scripting* ja que aquestes advertències de seguretat eviten que des d'una domini es pugui accedir a un recurs d'un altre domini, i així injectar un *script* maliciós en JS. Aquesta qüestió de seguretat és explicada en detall al punt 3.4.23.4.2. El servidor intermediari ha d'implementar mecanismes per assegurar-se de que les dades depositades per una aplicació nativa siguin transmeses a l'aplicació Javascript.
- Opcionalment, es pot filtrar per IP.

El codi Javascript sol·licita els resultats de l'operació de signatura al servidor intermedi, però per garantir que el servidor només retorna els resultats al client que les ha originat s'han de seguir certes recomanacions en quant a seguretat:

- El client ha de generar un identificador "al vol", per exemple un UUID (*Universally Unique Identifier*) segons el RFC 4122 [27], al iniciar la transacció i demanar les dades al servidor indicant aquest UUID.

- El client ha de generar "al vol" una clau única de xifrat simètric, per exemple 3DES [28], que es traslladarà en la invocació a l'aplicació nativa, i aquesta pujarà les dades xifrades al servidor.
- Un cop pujades les dades, aquestes han de tenir un cert "període de validesa", i quan hagin caducat seran esborrades.
- S'ha d'utilitzar el protocol SSL en totes les transaccions.

2.2.3. Implementació de l'aplicació nativa per ser invocada per protocol

Un avantatge de la invocació per protocol és que no ha de seguir una API específica ni una normativa concreta. D'aquesta manera, és possible realitzar una implementació Java que sigui útil per tots els sistemes operatius, ja que una aplicació JSE funciona perfectament en Windows, Linux y OS X.

Associació del protocol

Segons el sistema operatiu s'associarà d'una forma o d'una altra el protocol a l'aplicació:

- Microsoft Windows:
  - Associació en el Registre de Windows. S'ha de realitzar mitjançant un programa específic d'instal·lació.
- Apple OS X:
  - Declarat en l'aplicació.
- Linux:
  - Associació dual en Firefox i en el gestor de finestres del sistema operatiu (*Gnome*, *KDE*,...). S'ha de realitzar mitjançant un programa específic d'instal·lació, per exemple un paquet DEB.

Dependència del JRE per aplicacions desenvolupades en Java

Òbviament, qualsevol aplicació Java necessita el JRE per executar-se. Per tant, l'usuari no només ha de tenir l'aplicació instal·lada i actualitzada, sinó que també ha de tenir el JRE.

Aquest manteniment pot comportar que aplicacions instal·lades anteriorment per l'usuari, al instal·lar el nou JRE compatible amb l'aplicació nativa, no siguin compatibles amb altres aplicacions del seu equip.

Per política de publicació d'aplicacions d'Apple a la seva AppStore, no està permès la dependència externa del JRE i això inhabilitaria les actualitzacions automàtiques als usuaris de OS X.

Per solucionar aquests problemes, és recomanable que l'aplicació inclogui internament un JRE pel seu ús exclusiu. Al ser un JRE no registrat en el sistema operatiu no influiria en els JRE ja existents ni en la configuració de l'usuari, al no instal·lar ni necessitar un *Java plugin* no introdueix problemes de seguretat en

l'equip i al ser un JRE seleccionat i testejat quedaria garantida la compatibilitat amb l'aplicació nativa.

2.2.4. Conclusions

L'experiència de l'usuari en la invocació per protocol té un escull important quan l'usuari no té l'aplicació instal·lada. Les aplicacions Javascript no poden, per motius de seguretat, saber quines aplicacions hi ha instal·lades a l'equip i, per tant, invocaran el protocol "a cegues", suposant que l'aplicació està instal·lada, provocant un error. S'ha de tractar aquest error de forma que l'usuari no quedi desconcertat, i avisar-lo de que necessita l'aplicació local abans d'iniciar el procés.

Un altre problema és el ja mencionat canvi de context, que sol ser molest i fa que l'usuari hagi de tornar manualment al navegador.

Finalment, un altre tema a valorar és la necessitat d'un servidor intermedi, el qual fa necessari un petit servei web a més del servidor web amb l'aplicació web. Aquest servidor intermedi fa a més que hi hagi més connexions de xarxa i a conseqüència més tràfic de dades, i per tant alenteix l'operació i augmenta la dificultat de localitzar els possibles errors en el procés de signatura.

2.3. Comunicació per servei

La major deficiència de la invocació per protocol és la necessitat d'un servidor intermedi que gestioni la comunicació de tornada al navegador. No obstant, és possible que una aplicació nativa actuï com a servidor de *sockets* TCP (*Transmission Control Protocol*) o de servidor HTTP i disposar d'una aplicació Javascript en un navegador que estableixi connexions amb la mateixa, tal i com es mostra a la Fig. 25.

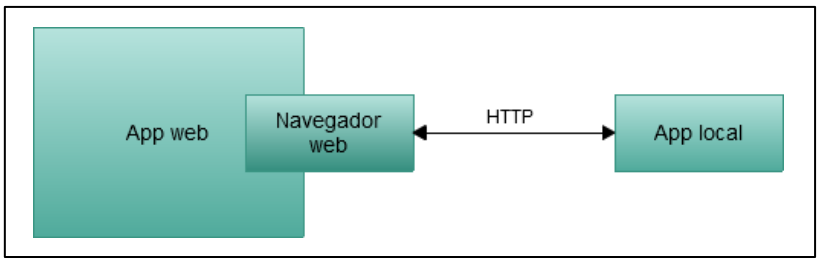

Fig. 25 Esquema bàsic de la comunicació per servei

L'aplicació local s'executaria com un servei en la màquina de l'usuari, escoltant a l'espera d'una sol·licitud en un port TCP fixe, mentre que l'aplicació Javascript de la signatura és la que inicia el diàleg mitjançant una crida *WebSocket o HTTP* al *localhost*.

2.3.1. Compatibilitat, experiència d'usuari i altres consideracions

Compatibilitat

Parlant en termes de compatibilitat, tant Windows, com OS X i Linux, permeten realitzar invocacions per servei.

Experiència d'usuari

Utilitzant un servei local, al no haver temps d'espera per arrencar l'aplicació ni diàlegs d'advertència, l'experiència de l'usuari és més gratificant. No obstant, segueix havent aquests possibles canvis de context que poden desconcertar a l'usuari i que no existien amb els *applets*.

Especificacions per l'aprofitament d'un servei

Al tractar-se d'un proveïdor de serveis a nivell de sistema operatiu, és pràcticament obligatori desenvolupar una interfície nativa en cada sistema operatiu seguint les especificacions del fabricant, i així s'obtenen els beneficis de que disposa un servei típic de cada sistema operatiu:

- Gestió a nivell de sistema operatiu, ja que diferents processos poden ser instàncies d'un mateix programa, d'aquesta manera deixem que el sistema operatiu gestioni una única instancia.
- Autoarrencada en cas de caiguda.
- Configuració comuna, com per exemple permisos d'administrador.

Segons el sistema operatiu s'ha d'aplicar els requisits definits en les seves especificacions, tal i com es mostra a la Taula 2.5

Taula 2.5 Especificacions per l'aprofitament d'un servei segons el sistema operatiu

Windows	Requisits definits a les especificacions "*Services Control Manager"* de Microsoft. A partir de les restriccions de seguretat imposades des de l'aparició de Windows Vista, és necessari seguir les pautes de "*Windows Service Hardening"*. API disponible en C++ i .NET.
OS X	S'ha d'implementar les interfícies definides per Apple. API disponible en Objective C.
Linux	Hi ha diferents opcions. Una d'elles es desenvolupar-la en Java i arrancar-la a partir d'un script del sistema (SH, BASH,...).

Per tant, podem observar que serien necessaris diferents desenvolupaments natius per abastar tots els sistemes operatius.

Sense tràfic extern

En aquest cas, la sincronització del navegador i l'aplicació a través d'un servei no té tràfic de xarxa externa, i per tant, ens estalviem:

- Consum de xarxa
- Temps de transmissió
    - Les dades s'envien des del navegador al servei extern (temps d'espera en el navegador web).
    - Les dades es descarreguen des del servei a l'aplicació (temps d'espera en l'aplicació).
    - La signatura s'envia al servei extern des de l'aplicació (temps d'espera en l'aplicació).
    - La signatura es descarrega des del servei extern al navegador (temps d'espera en el navegador web).

Aquests costos s'incrementarien de forma proporcional a la mida de les dades.

Seguretat

La seguretat és la part pendent d'aquesta infraestructura per diferents motius:

- Un servei accessible des de TCP és un punt d'entrada molt sensible a les claus privades i certificats:
    - La necessitat d'utilitzar sempre el mateix port fa que no sigui necessari una exploració prèvia de ports per realitzar un primer atac.
    - Encara que es configuri per només acceptar connexions locals, és fàcil impostar la IP d'origen per a que sembli una local.
    - Qualsevol servei TCP és susceptible d'atacs. Amb facilitat pot haver compromisos de seguretat en el codi: tractament de buffers, inseguretats pròpies de l'entorn d'execució (Visual C++, Java, .NET...).
- En entorns de terminal, s'introdueix una nova amenaça, la suplantació d'identitat d'usuari, que, com ja s'ha comentat anteriorment, derivaria en l'accés a les claus privades i certificats d'un altre usuari dins de la mateixa xarxa de terminals.

Xifrat de dades transmeses

Ja que les transmissions es realitzen a través de connexions *HTTP*, si no s'utilitza un xifrat adequat de dades, un senzill anàlisi de xarxa exposaria tota la informació.

2.4. Invocació per protocol i comunicació per servei

La comunicació per servei eliminava el gran problema de la invocació per protocol, que és la necessitat d'un servidor intermedi per la transmissió bidireccional contra l'aplicació web, però introduïa una sèrie de problemes que la feien pràcticament inviable.

No obstant, és possible plantejar una combinació de les dues opcions per solucionar limitacions d'ambdues sense prescindir dels avantatges de cap d'elles. Es podria plantejar un proveïdor de servei per port TCP que fos arrancat mitjançant una invocació per protocol, passant-li els paràmetres necessaris sobre la prestació del servei en aquella invocació.

2.4.1. Problemàtica resolta amb la combinació

- La comunicació per servei utilitza sempre el mateix port TCP, el que fa que es puguin planificar els atacs.
    - L'aplicació Javascript pot determinar de forma aleatòria una sèrie de ports com a candidats per ser utilitzats pel servei. Aquesta llista s'entregaria com un paràmetre en la invocació del protocol. Així, el port sempre seria diferent.
- La comunicació per servei sempre deixa un port obert, esperant noves invocacions, lo que el fa una exposició a atacs permanents.
    - El servei es podria iniciar només mitjançant una invocació per protocol, i tancant-se al acabar la petició. Així, la "vulnerabilitat" de permetre a una connexió TCP accedir al repositori no està sempre activa i es redueix el temps en el qual seria vulnerable.
- La comunicació per servei podria admetre peticions d'atacants que haguessin falsejat la IP o la identitat.
    - La invocació per protocol pot indicar un número de petició (UUID generat aleatòriament), i el servei admetre només peticions que coneguin aquell UUID. Com la invocació per protocol no es susceptible a escoltes, ni a atacs "*man in the middle"*, no seria possible impostar l'origen de les comunicacions.
- La comunicació per servei, al utilitzar un número fixe de port TCP, causa conflictes en servidors de terminal, on totes les sessions comparteixen origen de xarxa.
    - Com el servei rep una llista aleatòria de ports TPC a utilitzar, es poden anar testejant en ordre fins que es trobi un port de la llista lliure. Al tenir la llista un milers de ports, normalment no assignats, fa que el port d'un altre sessió només estarà ocupat durant el temps que necessiti per realitzar la lògica de firma i quedant lliure al finalitzar.
- En la comunicació per servei, l'intercanvi de claus per l'establiment de xifrat de les comunicacions queda exposat a atacs de "*man in the middle"*.
    - Ara es pot realitzar aquest intercanvi amb claus com paràmetres en la invocació del protocol, el qual no és vulnerable per no ser una comunicació de xarxa, sinó un procés local del sistema operatiu.

Amb una combinació dels dos s'obtenen els avantatges dels dos i es supleixen les carències que tenen per separat. En la Fig. 26 s'exposa un esquema aclaridor del *workflow* que seguiria.

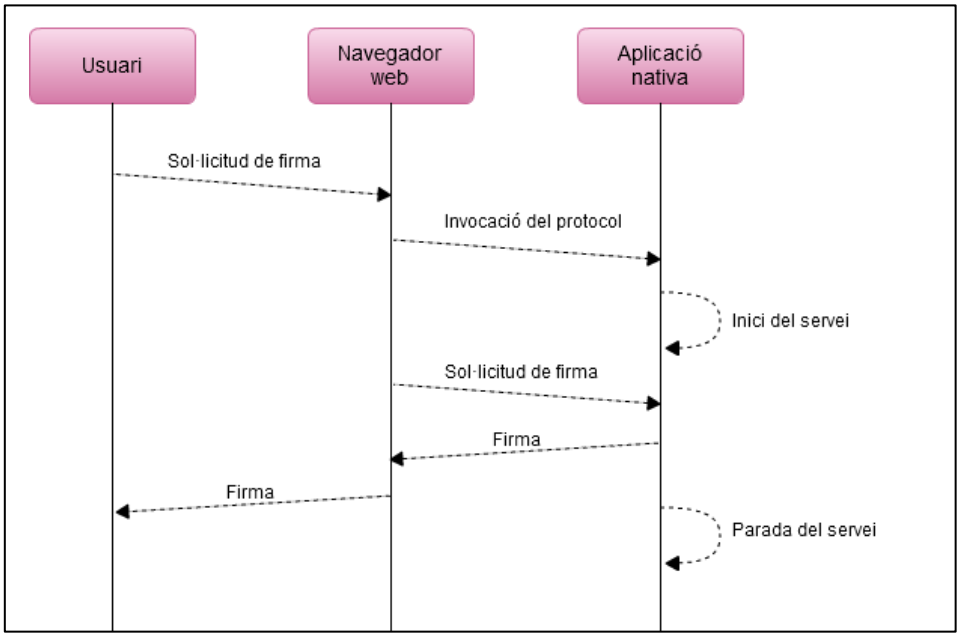

Fig. 26 Descripció funcional de la seqüència

2.4.2. Consideracions d'implementació

Compatibilitat en sistemes operatius

La combinació aquesta hereta de forma general la compatibilitat de la invocació per protocol, per tant, no hi ha cap problema en aquest camp.

Compatibilitat segons navegador web

Tots els navegadors accepten invocació de protocol excepte Windows 8 en UI Modern, per tant, segueix essent compatible.

Espera des de la invocació per protocol fins la plena operativitat del servei

Un dels principals desavantatges d'aquesta estratègia es troba en que després de la invocació per protocol per part del Javascript a l'aplicació nativa no és possible iniciar immediatament el diàleg amb el servei, ja que triga uns segons en estar operatiu.

Ja que no és possible determinar el temps d'espera perquè depèn de masses factors com per exemple, la versió de l'entorn d'execució Java, la potencia de la màquina, si ja existeix una instancia en execució, el sistema operatiu..., és necessari realitzar peticions a intervals determinats fins que una d'aquestes sigui atesa.

Aquesta lògica comporta certes restriccions:

- El servei s'ha de configurar per atendre una única petició, i rebutjar les altres.
- Les crides des de Javascript s'han de fer asíncronament, de forma que no hagi que esperar al *timeout* d'una petició per executar la següent.
- Un cop la petició és atesa, s'han de cancel·lar la resta.
- El temps d'espera entre peticions s'ha de calcular en base a un temps raonable en el que la petició és atesa.

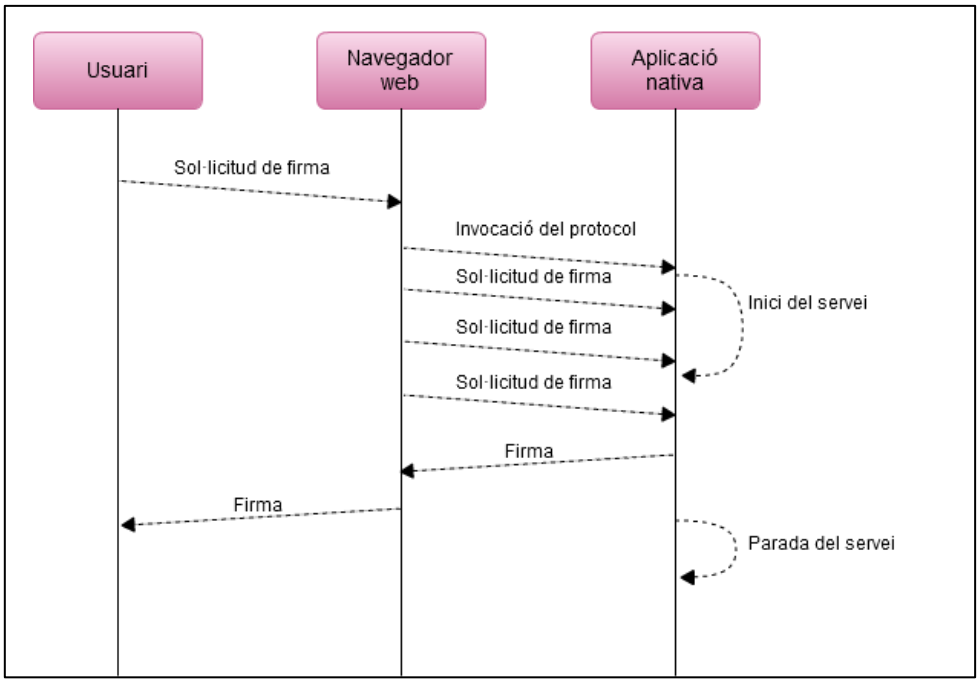

Fig. 27 Esquema funcional amb repetició de crides al servidor

2.4.3. Conclusió

El mix entre la invocació per protocol i la comunicació per servei, tal com ha quedat exposat, aporta avantatges interessants, amb un menor nombre de comunicacions de xarxa, un procés on tot te caire local i sense exposició externa de les dades de firma.

D'altra banda, cadascuna de les plataformes requeriria el seu propi medi de distribució de les aplicacions, l'AppStore, Google Play, DEB, o un instal·lador a mida per Windows.

2.5. API totalment Javascript

Essent un procés web, lo desitjable seria poder implementar una solució totalment en Javascript a partir d'una API per a que tingués màxima compatibilitat amb tots els navegadors web independentment del sistema operatiu.

Per desgracia, pel cas de la firma electrònica, no existeix ninguna API. Hi ha algunes iniciatives en progrés, que un cop finalitzades i adoptades per l'industria, podrien cobrir les necessitats d'aquest tipus d'aplicacions.

2.5.1. WebCrypto de W3C

El W3C treballa actualment en dues iniciatives per normalitzar una API Javascript de criptografia per navegador. Dins d'aquesta iniciativa hi ha dos grups:

- WebCrypto Key Discovery:
  - API per l'ús segur de claus privades des del navegador web.
  - Treballs liderats per Netflix.
- Web Cryptography API:
  - API per la realització d'operacions criptogràfiques, com RSA, des de Javascript.
  - Treballs liderats per Google i Netflix.

El segon grup de treball WebCryptography API no suposa un medi que permeti tenir aplicacions de firma 100% Javascript, ja que sense el concurs de claus privades no es possible realitzar firmes.

En canvi, el primer, la WebCrypto Key Discovery si és el punt crític que permetria aplicacions en firma Javascript, inclús si Web Cryptography API no estigues disponible, ja que hi ha biblioteques lliures per fer aquestes operacions.

Desgraciadament, WebCrypto Key Discovery és una especificació sobre la que quasi no es treballa i per tant, Netflix, principal patrocinador orienta clarament a la DRM (*Digital Rights Managements),* lo que pot dificultar la seva adopció generalitzada per la industria:

- Google (Chrome) basa el seu negoci en actius de lliure accés amb publicitat i s'ha manifestat en contra d'aquesta API.
- Mozilla (Firefox) no accepta ninguna iniciativa de control tipus DRM per la web.

D'aquesta manera és bastant improbable tenir disponible a curt o mig termini una API normalitzada per W3C i adoptat de forma generalitzada per l'industria que permeti desenvolupar aplicacions de firma 100% Javascript.

2.5.2. Aliança FIDO

Per un altre banda, FIDO (*Fast IDentity Online*) la qual és, un grup de fabricants, que treballa en una API pensada per una autenticació forta en entorns web mitjançant criptografia de clau publica, orientat principalment a pagaments en línia.

Entre els patrocinadors i promotors de l'aliança es troben Samsung, Microsoft, ARM i Google.

En les especificacions es contempla la realització de firmes electròniques, la seva orientació a la autenticació fa que no s'exposi una API PKCS#1 [29] de forma directa, per lo que el seu ús per realitzar firmes electròniques, ara per ara, no és possible.

2.5.3. Consideracions

Actualment no és possible una solució de signatura 100% JS, i no ho serà a curt o mig termini com una solució general per totes les plataformes. Igualment, veient la velocitat d'avanç de les tecnologies, no és raonable esperar que les API aquí descrites siguin finalment adoptades en cinc o més anys, si es que s'arribés a adoptar alguna.

Per tant, es pot dir que seria la opció ideal, però no haurà aplicacions construïdes amb aquesta tecnologia fins un cert temps.

2.6. Pros i contres de cada tecnologia

A continuació s'exposa els avantatges i desavantatges de totes les tecnologies:

- Complements per navegador Web
  - ✓ El seu desenvolupament és actualment viable.
  - × Incompatibilitat amb Windows 8 Modern UI.
  - × Requereix desenvolupaments específics per cada navegador en diferents tecnologies.
  - × Introdueix un factor d'inseguretat per permetre l'execució de codi privilegiat des de Javascript.

- Invocació per protocol
  - ✓ El seu desenvolupament és actualment viable.
  - ✓ Compatibilitat amb pràcticament tots els entorns.
  - × Requereix la instal·lació d'una aplicació local per part de l'usuari.
  - × Introdueix un factor d'inseguretat per requerir l'ús d'un servidor extern per la sincronització de dades amb el navegador.

- Connexió per servei
  - ✓ El seu desenvolupament és actualment viable.
  - × Requereix desenvolupaments específics per Windows, Linux, Mac OS X, ja que en cada sistema operatiu canvia la configuració d'un servei.
  - × Requereix la instal·lació d'una aplicació local per part de l'usuari.
  - ✓ Sense problemes de seguretat fàcilment explotables ja que s'elimina la necessitat d'un servidor extern i el servei està actiu durant un temps determinat.

- API Javascript
  - × El seu desenvolupament és actualment inviable.
  - ✓ Una única implementació per tots els entorns.
  - ✓ No requereix que l'usuari instal·li cap aplicació
  - ✓ Sense problemes de seguretat fàcilment explotables ja que s'executa tot en local.

2.7. Arquitectura escollida

Un cop vistes i valorades les diferents tecnologies per substituir l'*applet*, actualment i amb el software que té l'empresa, s'ha optat per implementar la solució de la comunicació per servei, ja que l'empresa disposa d'una aplicació d'escriptori en Java pròpia per realitzar signatures anomenada ClickSign. Per tant, la intenció és enllaçar l'aplicació web Portasigma amb l'aplicació ClickSign, que al ser en àmbit local, instal·lat a la màquina del propi usuari, si té permisos per accedir al repositori de claus i certificats. L'aplicació ClickSign serà iniciada des d'un servei de Windows, i s'iniciarà un servidor HTTPS al *localhost* per a que navegador i ClickSign puguin interactuar.

S'ha optat per aquesta tecnologia ja que segons els pros i els contres abans mencionats, és la solució amb més viabilitat de cara a un entorn multiplataforma i amb diferents sistemes operatius. D'aquesta forma s'estalvia la necessitat de crear una variant per a cada navegador web i versió d'aquest.

El fet que l'aplicació estigui en Java permet estalviar un desenvolupament per a cada sistema operatiu, de manera que només cal realitzar una única versió i no cal retocar el codi font. L'inconvenient que té és la necessitat de requerir l'aplicació instal·lada a la màquina del client i l'usuari tindrà que prendre's la molèstia en instal·lar-la.

Des del punt de vista empresarial, és un punt a favor ja que es dona a conèixer un altre producte i a més se'ls proporciona una aplicació d'escriptori per signar documents PDF gratuïtament.

CAPÍTOL 3. IMPLEMENTACIÓ DE L'ARQUITECTURA DE COMUNICACIÓ PER SERVEI

L'arquitectura escollida ha estat la de comunicació per servei (Fig. 28).

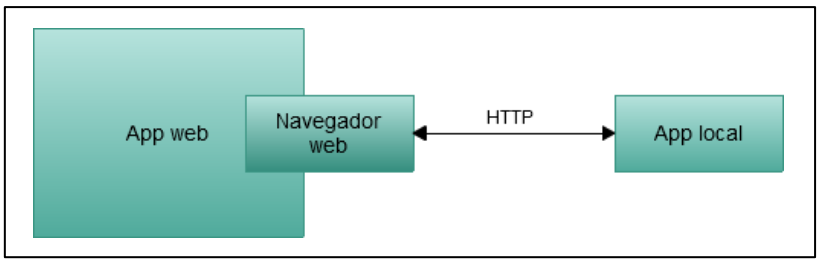

Fig. 28 Comunicació per servei

Com ja s'ha explicat en l'apartat 2.3, aquesta nova implementació també té algun inconvenient:

- Requereix desenvolupaments específics per Windows, Linux, Mac OS X, ja que en cada sistema operatiu canvia la configuració d'un servei.
- Requereix la instal·lació d'una aplicació local per part de l'usuari.

Però, ja que CS amb l'opció dels menús contextuals només funciona amb Windows, aquesta nova funcionalitat solament estarà disponible en aquest sistema operatiu.

Aquesta implementació implica tenir un servidor que permeti connectar navegador web amb l'aplicació local. No obstant, al tenir una aplicació local instal·lada a la màquina del client es pot iniciar un servidor en la seva màquina i així obrir un canal de comunicació entre el navegador web i la màquina del usuari. En el cas de la implementació entre PS i CS, aquest últim realitza de servidor, i a través de crides HTTP generades al codi JS del navegador, PS es pot intercanviar informació amb CS el qual pot accedir als certificats de l'usuari.

En aquest capítol s'explicarà la implementació del nou sistema desenvolupat a l'aplicació PS per a poder realitzar la identificació i la signatura amb el certificat digital, basat en l'arquitectura escollida: comunicació per servei.

En primer lloc es descriurà de forma general el procés que s'ha seguit per arribar al funcionament de les dues aplicacions conjuntes i els canvis que s'han anat introduint a mesura que s'implementava. A continuació s'explicarà el mètode d'autenticació de la solució i el procés de signatura i, per últim, les modificacions que s'han realitzat a l'aplicació CS.

3.1. Descripció general

Primerament, pel que fa a l'arquitectura, es va decidir optar per no tenir el servidor corrent sempre. Per contra, quan s'entra a l'aplicació PS a través de codi Javascript, es comprova si el servidor està escoltant i, en cas negatiu, s'intenta executar-lo a partir d'un *protocol handler* i després es segueix amb el *workflow* habitual.

Aquesta lògica es va arribar a desenvolupar, però l'inconvenient que tenia era que el servidor trigava aproximadament uns 30 segons en arrencar, configurar tots els paràmetres criptogràfics i realitzar la corresponent lògica d'obtenir els certificats o signar. Donat que es tracta d'un temps massa gran que no permet oferir al client una experiència correcta es va desestimar. Per aquest motiu, es va decidir implementar un servei per encendre el servidor conjuntament amb el sistema operatiu. D'aquesta manera es guanya velocitat a l'hora d'utilitzar PS i simplicitat en el codi Javascript de forma que la càrrega computacional a la part del client també es redueix .

Un cop això va quedar decidit, abans de començar a realitzar modificacions al codi font, es va analitzar l'abast que tenia aquesta implementació, i es va decidir fragmentar-la en dues grans parts. Per una banda l'autenticació amb certificat digital i, per l'altra la signatura.

En primer lloc es va adreçar el problema de l'autenticació, fent ús de la forma més bàsica, sense mesures de seguretat extres. Es va desenvolupar el servidor HTTP a CS amb tres contextos: un per a obtenir els certificats, un per realitzar una signatura digital, i un com a feedback per saber si el servidor està encès.

Amb el servidor ja funcional, però en una versió molt bàsica i sense optimitzar, es va passar a integrar l'aplicació web amb el servidor en la part de l'obtenció dels certificats, modificant el codi Javascript.

Al aconseguir llistar els certificats, el següent pas va ser identificar-se. Això inclou un procés de signatura d'uns caràcters clau anomenat *challenge* per saber realment que el certificat amb el que es vol identificar està instal·lat al sistema operatiu del client, o a un dispositiu criptogràfic. Pel fet d'haver de signar aquest *challenge*, al servidor es va desenvolupar el codi per a realitzar la signatura en el seu context corresponent. Més endavant en aquest mateix capítol s'explicarà en detall.

En aquest punt ja està completat el procés d'autenticació a l'aplicació amb CS. A continuació, es substitueix el codi Javascript realitzat per a la signatura amb *applet* pel de signatura amb CS. Com que PS té dues maneres de signar, la signatura d'un únic document i la múltiple, al ser dos fluxos de codi diferents, cal integrar els dos.

A partir d'aquí, ja s'ha aconseguit una primera versió totalment funcional del nou sistema de signatura. El següents passos aniran encaminats a afegir seguretat, donar estabilitat a la solució, donar *feedback* dels possibles errors i a millorar el rendiment.

Referent al tema de la seguretat, s'ha afegit un *token* de seguretat, el qual permet o no que es realitzi la signatura de les dades des de CS. També s'ha modificat el servidor de CS, per a que en comptes de treballar amb HTTP, treballi amb HTTPS i no es pugui fer un *sniffing* del tràfic de tal forma que es vegi com connectar amb el servidor local.

Per a donar un *feedback* a l'usuari final, si CS té un error signant les dades, s'informa a PS. Addicionalment s'ha afegit un sistema de *logs* a CS on queden escrits en un fitxer les traces dels errors que sorgeixin.

Finalment, ja fora del context del codi font, s'ha modificat el registre de Windows per a que s'iniciï en forma de servei, s'ha autoinstal·lat un certificat amb una CA d'Isigma creada específicament per a la connexió SSL de CS, s'han donat permisos d'administrador a la carpeta arrel de CS per evitar problemes d'execució i s'ha afegit la nova opció al menú contextual de CS per encendre el *plugin web*, entre d'altres tasques totes automatitzades en el moment de la instal·lació.

3.2. Autenticació

El procés d'autenticació a PS amb certificat digital implementat es mostra a la Fig. 29.

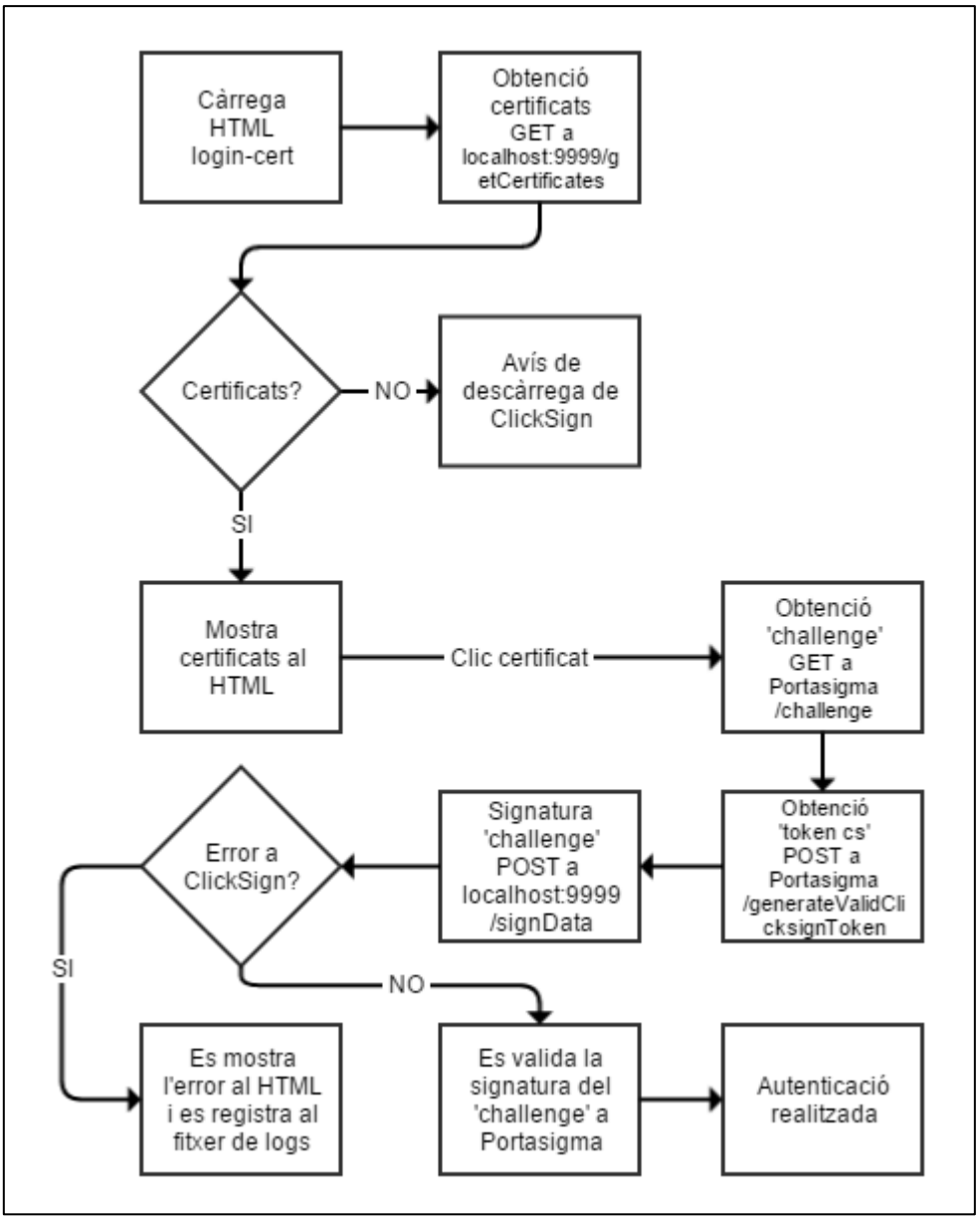

Fig. 29 Esquema del procés d'autenticació

Al carregar la pàgina web inicial de PS anomenada *login-cert*, s'executa el codi JS que realitza la crida des del navegador cap al servidor de CS per a obtenir els certificats. A la Fig. 30 es veu el missatge d'espera mentre s'obtenen els certificats.

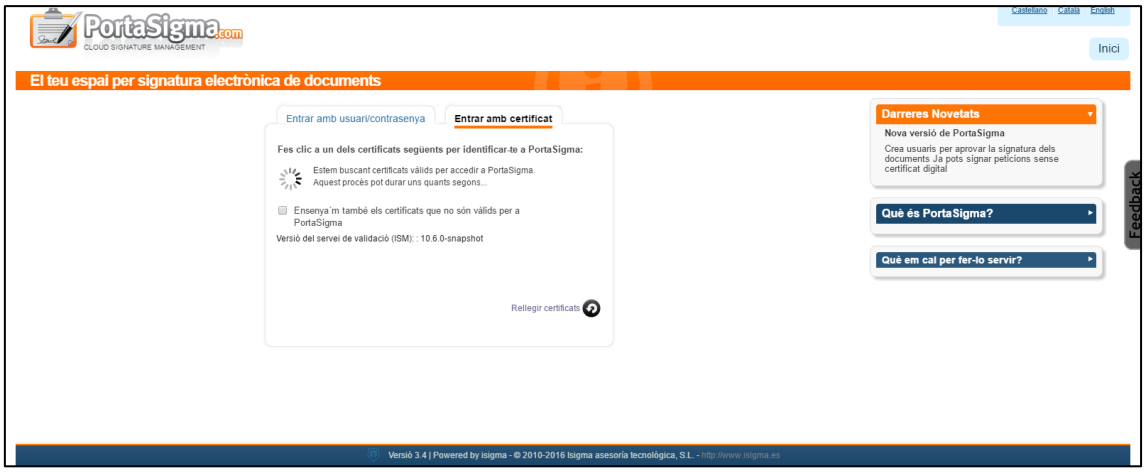

Fig. 30 Pàgina *login-cert* mentre s'obtenen els certificats

Un cop s'ha rebut la resposta del servidor, des del navegador amb el codi JS es comprova que el missatge rebut sigui els certificats i no un codi amb un error produït a CS. Els certificats, que es reben amb codificació base 64 i separats per un punt i coma, es tracten i es validen mitjançant OCSP, s'afegeixen a la llista corresponent de vàlids o invàlids i es mostren a la web de PS des del navegador com es pot veure en la Fig. 31.

Fig. 31 Certificats vàlids i invàlids a PS

En cas de que la petició al servidor de CS realitzada al carregar *login-cert* no sigui atesa, a la pàgina de PS apareix a la part superior un avís groc amb uns missatges on s'indica que possiblement no estigui corrent CS en la màquina del propi usuari (Fig. 32). Aquest avís conté un botó per descarregar CS i un enllaç amb el títol "Què és ClickSign?" que porta a la web del producte.

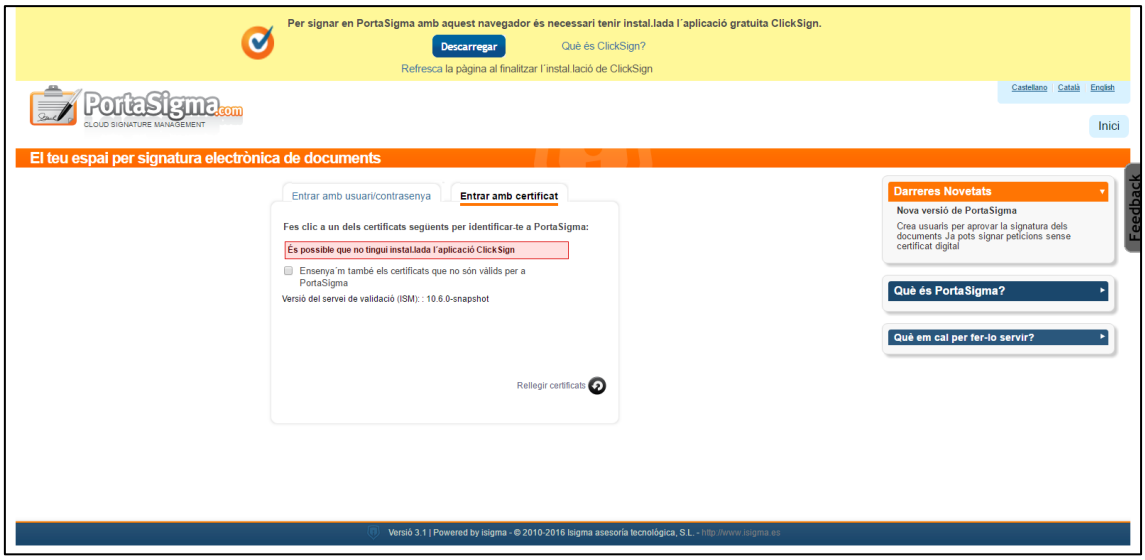

Fig. 32 Avís de com descarregar CS

Tornant al cas favorable en el qual està CS instal·lat, al realitzar el clic al certificat, es desencadena un esdeveniment JS que selecciona les dades del certificat i realitza a partir d'una crida HTTP amb mètode GET [30] a https://app.portasigma/challenge l'obtenció del *challenge*. Aquest *challenge* és l'identificador de la sessió HTTP amb el qual relaciona PS a aquest usuari i més tard s'utilitzarà per completar l'autenticació.

Després d'obtenir el *challenge,* es realitza l'obtenció amb una crida HTTP amb mètode POST [30] a https://app.portasigma.com/generateValidClickSignToken del *token* que s'utilitza per distingir qui realitza la crida a CS, si el *token* no està generat des de PS, CS ho detecta i retorna un codi d'error. I finalment, després d'obtenir el *challenge* i el *token* es realitza la crida HTTP de signatura, des del codi JS de la web de PS, a CS passant el *challenge* com a dades a signar, i el *token,* a més del certificat que es vol utilitzar i un seguit de paràmetres de funcionament que en l'apartat de CS es detallen.

Un cop s'ha fet la crida a CS, aquest comprova a través de les mesures de seguretat instaurades, com el *token* i algun altra, explicades més en detall al punt 3.4.1., que pot signar el *challenge* i retornar-lo. PS verifica la signatura del *challenge* i es completa l'autenticació. Si per alguna raó, a CS succeís algun error, retornaria el corresponent codi d'error en comptes de les dades signades, i a PS es mostraria l'error.

3.3. Signatura

El procés de signatura, és un punt crític en el qual, s'ha d'afegir un extra de seguretat ja que comporta responsabilitats a nivell legal. Per tant, per evitar que qualsevol crida fos acceptada pel servidor i signés les dades introduïdes, s'ha afegit un sistema de seguretat de tres fases, que permet distingir si és PS qui realitza la crida o és un altre sistema. En la Fig. 33 es mostra un diagrama amb la casuística de com es comporta el servidor en funció de si s'envien correctament els paràmetres que demana per signar.

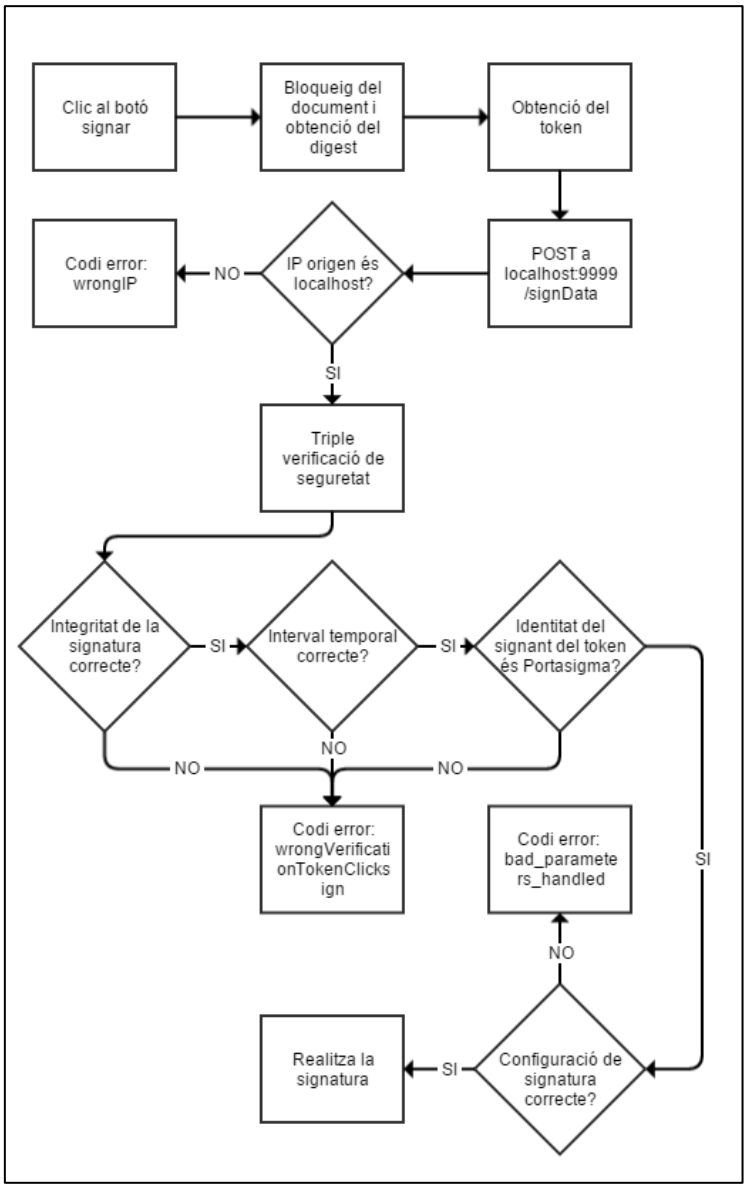

Fig. 33 Procediment de signatura

El procés comença amb el clic al botó de signatura (Fig. 34), aquest desencadena un esdeveniment en JS que realitza una crida HTTP amb mètode GET cap a PS que bloqueja el document actual i obté el *digest* com a resposta, a continuació es captura la data des de la màquina de l'usuari a través de JS i s'envia a signar per PS via HTTP amb el mètode POST, aquesta data amb la signatura en format PKCS7-*attached* és l'anomenat *token*, que més endavant en aquest mateix punt es detallarà el seu ús.

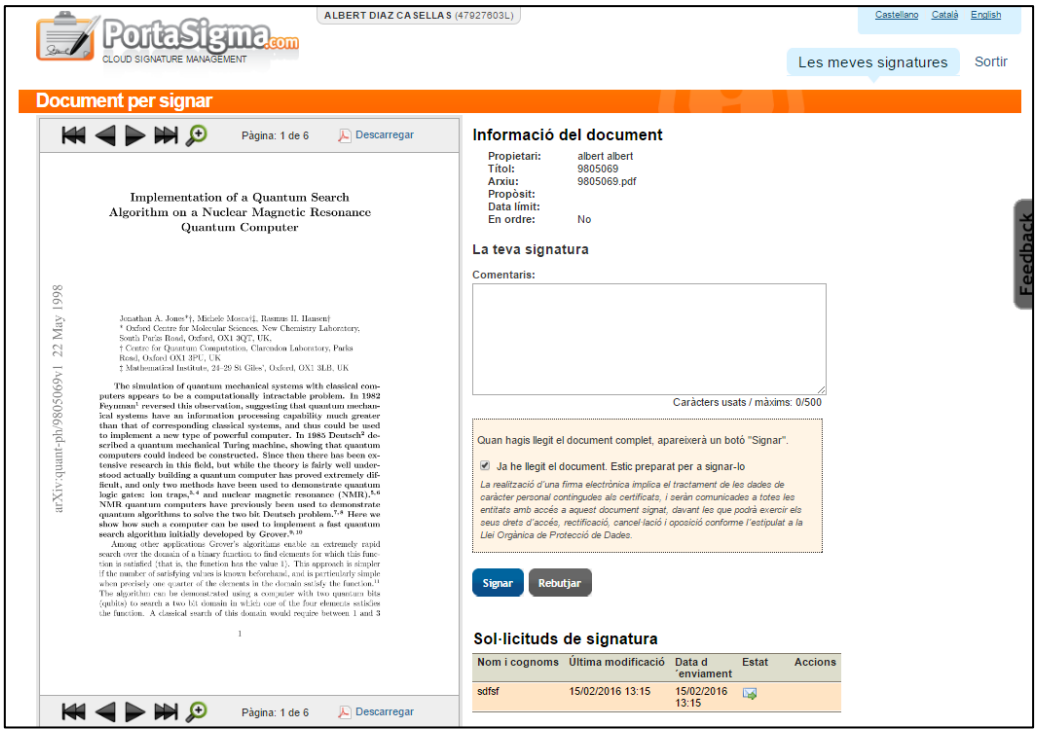

Fig. 34 Pantalla de signatura

Un cop es té el document bloquejat, l'identificador del document i el *token*, es realitza un altra crida HTTP POST a CS a la direcció https://localhost:9999/signData amb la resta de paràmetres:

- *challenge:* són les dades que es volen signar
- *base:* la base de codificació, base 64
- *algorithm:* l'algoritme de xifrat del *hash*, SHA1
- *b64Cert:* certificat codificat en base 64
- *signOption:* opció de signatura, si 0 → simple; si 1→ múltiple
- *totSigns:* número total de signatures, si *signOption* és 1, aquest nombre serà major que 1.
- *remSigns:* signatures que falten respecte del camp *totSigns*, si *signOption* és 1, aquest nombre serà igual o major a 1.
- *token:* és el camp on s'envia el *token* de seguretat que crea PS i el signa agafant la data des de Javascript.

El servidor de CS, al rebre la petició HTTP, la primera comprovació que realitza és la de comparar la IP origen amb la de *localhost*. Si la petició no prové d'aquesta direcció serà rebutjada i es contestarà amb el codi *wrongIP*, en cas favorable es realitza la triple verificació per assegurar que l'origen és PS. Aquesta verificació consisteix en tres passes:

1. **Comprovació de la integritat de les dades.** Es comprova que les dades es poden desencriptar amb la clau pública que s'obté de la pròpia signatura PKCS7-*attached*. D'aquesta forma s'assegura que no s'han modificat un cop han estat signades per la clau privada de PS.
2. **Finestra temporal.** Amb la data obtinguda, es realitza una comparació d'aquesta amb un interval de temps d'un minut. D'aquesta forma

s'assegura que el *token* tingui una caducitat i no pugui ser reutilitzat infinitament.

3. **Comprovació del signant.** Es compara certificat obtingut de la signatura PKCS7 amb el certificat de PS carregat a CS. Amb aquesta verificació, s'aconsegueix assegurar que el signant és PS.

Si alguna d'aquestes comprovacions ha sortit desfavorable, es retorna el codi d'error *wrongVerificationTokenClicksign*, i en el cas que la triple comprovació sigui correcta, es procedeix a verificar que els paràmetres de configuració de signatura són correctes. Aquests paràmetres són *signOption, remSigns, totSigns* i varien els seus valors en funció de la signatura simple o múltiple.

Signatura simple

- *signOption:* pren el valor 0 per a aquest cas.
- *remSigns:* prové de *remaining signatures*, i indica les signatures que queden per signar, en aquest cas sempre serà 1.
- *totSigns:* indica el número de signatures totals que es realitzaran, en el cas de la signatura simple, prendrà el valor 1.

Si en aquest cas, els paràmetres no prenen els valors explicats, CS retornarà com a resposta a la petició HTTP l'error *bad_parameters_handled*.

Signatura Multiple

- *signOption:* pren el valor 1.
- *remSigns:* sempre serà igual o major que 1.
- *totSigns:* en el cas de la signatura múltiple, prendrà un valor major que 1 i si per algun cas, *totSigns* fos menor que *remSigns*, CS no permetria la signatura.

Al igual que en la signatura simple, si hi hagués algun error en els paràmetres, retornaria el mateix codi d'error, *bad_parameters_handled*.

3.4. Modificacions a ClickSign

En aquest punt del capítol 3, es detallaran totes les modificacions realitzades a CS, tant a nivell de codi amb el desenvolupament del servidor, com a canvis al menú contextual per encendre'l, el procés d'instal·lació del SSL, etc. I es realitzarà una valoració del nivell de compatibilitat que té aquesta solució un cop ha estat realitzada en les diferents versions de Windows, des de XP fins a W10.

3.4.1. Desenvolupament del servidor

A CS s'ha creat una classe anomenada *LinkToPortasigma.java*, on està desenvolupat tot el servidor HTTPS a partir de la classe Java HttpsServer [32].

El servidor consta de tres contexts [33] on accedir per realitzar les diferents opcions:

- getCertificates

Per accedir a aquest context no cal passar cap paràmetre de seguretat, ja que només obté els certificats de la maquina del client, sense la clau privada, per tant, no és ningun perill que algú el pugui obtenir.

Per incloure la lògica que obté els certificats des del sistema operatiu o des del dispositiu criptogràfic, s'ha aprofitat el codi que ja conté l'ISM. Solament, s'ha hagut de realitzar modificacions a nivell del tractament d'errors, ja que en aquest cas, el *feedback* es vol tenir a la pàgina de PS i per això s'ha hagut de modificar les excepcions que treia el codi, cap a missatges d'errors que es retornen a PS, i a més un registre a un fitxer de la màquina local, utilitzant el sistema de *logs* amb la traça del error i la data, per tenir més detalls.

- signData

Al cridar aquest context, es realitzarà la signatura de les dades que se li hagin passat. Per realitzar-ho, se li han de passar uns arguments adequats, els quals són:

  - *Challenge:* són les dades que es volen signar
  - *Base:* la base de codificació, base 64
  - *Algorithm:* l'algoritme de xifrat del *hash*, SHA1
  - *B64Cert:* certificat codificat en base 64
  - *signOption:* opció de signatura, si 0 → simple; si 1→ múltiple
  - *totSigns:* número total de signatures, si és *signOption* és 1, aquest nombre serà major que 1.
  - *remSigns:* signatures que falten respecte del camp *totSigns*, si *signOption* és 1, aquest nombre serà igual o major a 1.
  - *Challengecs:* és el camp on s'envia el *token* de seguretat que crea PS i el signa agafant la data des de Javascript.

Per realitzar la lògica de la signatura, s'ha aprofitat l'ISM, però al igual que en el cas de l'obtenció de certificats, s'han modificat algunes excepcions i tractaments d'errors per utilitzar el sistema de *logs* i que els missatges es mostrin a l'aplicació web.

- ping

Aquest context només serveix per verificar que el servidor està encès. En el moment de cridar-lo tant sols retorna un missatge com que està corrent. A la Fig. 35 es veu aquesta contestació del servidor.

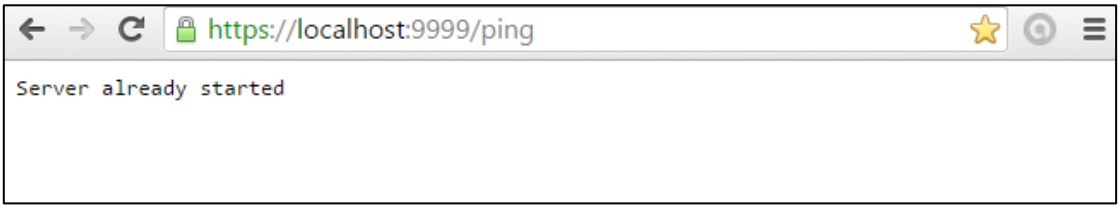

Fig. 35 Resposta del servidor de CS al context /ping

En el moment en el que el servidor s'inicia, a la barra de tasques de Windows, apareix la icona de CS (Fig. 36), per a que l'usuari tingui un *feedback* de que el *plugin* està corrent.

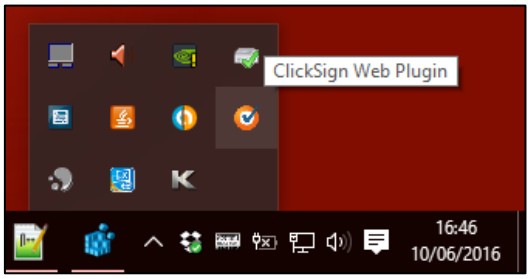

Fig. 36 Icona de CS a la barra de tasques

3.4.2. Altres modificacions

En aquest punt s'explicaran diferents modificacions que s'han realitzat per aconseguir un sistema de signatura segur i estable. A més, de que pel fet del funcionament de CS a través del menú contextual, s'ha hagut de afegir la nova opció d'encendre el servidor d'aquesta manera.

3.4.2.1. Inclusió de SSL

Per a augmentar el nivell de seguretat del servidor i evitar que algú faci un *sniffing* del tràfic de *localhost*, s'ha decidit afegir el protocol SSL al servidor de forma que tot el tràfic vagi xifrat.

Per fer-ho ha calgut prèviament crear una CA i un certificat amb propietats per ser usat per autenticar amb SSL, signat per aquesta CA.

Creació de la CA

Amb l'eina *KeyTool* [34] de Java, es genera un certificat junt amb la clau privada. Per això és necessari crear una *KeyStore* [35] per guardar el certificat i la seva clau privada, per tant serà un camp addicional a la comanda del *KeyTool*. En la comanda s'indicarà el CN en aquest cas com volem que reconegui el domini serà *localhost*, l'algoritme per generar les claus, RSA en aquest cas, s'indicarà que és un certificat de CA, això vol dir, que serà autosignat i el període de validesa, s'indica també que el certificat ha de permetre autenticar un servidor i un client, per això s'afegeixen els camps "*EKU=serverAuth,clientAuth"* que es refereix al *Extended Key Usage* [36], on a més de permetre l'autenticació com és el nostre cas, permet més usos.

```
keytool -genkey -noprompt -trustcacerts -keyalg RSA -alias localhost
-dname "CN=localhost, C=Barcelona, OU=Isigma" -keypass k3ypwd -
keystore keystore.jks -storepass 123456 -ext "BC=ca:true" -ext
"KU=dig,non,keyE" -ext "EKU=serverAuth,clientAuth" -validity 3650 -
keysize 2048
```

Fig. 37 Comanda per crear la CA amb l'eina *KeyTool*

Un cop executat aquesta comanda ja es té generat el certificat de CA (Fig. 38). El següent pas es exportar-lo amb extensió .cer per i afegir-lo a la carpeta on es troben tots els certificats de CA que reconeix CS.

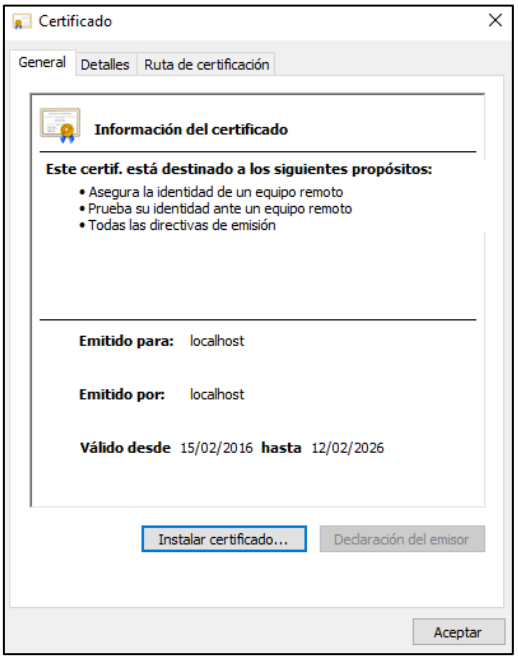

Fig. 38 Certificat d'autenticació SSL

Instal·lació del certificat al sistema operatiu

El següent pas es aconseguir que el sistema operatiu confiï en aquesta nova CA i no bloquegi la connexió al accedir-hi de forma que es tingui que afegir una excepció en el navegador des del qual s'està executant PS. Per fer-ho, es modifica l'instal·lador de CS, i se li afegeix la comanda mostrada en la Fig. 39 en llenguatge NSIS [37]:

```
Exec 'cmd /C "icacls "$INSTDIR" /q /c /t /grant %username%:(OI)(CI)F
& certutil -addstore -f "Root" "$INSTDIR\certs\ssl-cert.cer" &
certutil -A -n "$INSTDIR\certs\ssl-cert.cer""'
```

Fig. 39 Comanda NSS per instal·lar el certificat de CA al repositori

Aquesta comanda bàsicament el que fa és executar una consola amb permisos d'administrador i instal·lar el certificat al repositori de Windows [38].

Càrrega del certificat al servidor

En el moment d'iniciar el servidor, aquest agafa el certificat i la clau privada per xifrar d'un contenidor *Java Key Store*, desat en la carpeta arrel d'instal·lació, amb una clau per a que ningú pugui accedir directament i obtingui el certificat i la clau privada de CS.

3.4.2.2. Inici automàtic amb el sistema operatiu

Per evitar que l'usuari tingués que encendre el servidor cada cop que volgués utilitzar PS, s'ha modificat el registre de Windows per fer que s'encengui al arrancar amb el sistema operatiu.

Per fer això, prèviament s'ha creat un fitxer amb l'extensió *bat* anomenat *plugin-ps.bat* que s'encarrega d'executar la classe corresponent per encendre el servidor. El que es fa és afegir al registre l'execució d'aquest fitxer.

Aquesta entrada es crea entrant la següent comanda (Fig. 40) al fitxer NSIS de l'instal·lador [39]:

```
WriteRegStr HKCU 'Software\Microsoft\Windows\CurrentVersion\Run' 'clicksign'
 '"$INSTDIR\menu\hstart.exe" /noconsole "$INSTDIR\menu\plugin-ps.bat"'
```

Fig. 40 Comanda NSS per iniciar automàticament el servidor de CS

'clicksign' és el nom que pren la variable, i "$INSTDIR\menu\hstart.exe" /noconsole "$INSTDIR\menu\plugin-ps.bat" executa l'executable *hstart* passant-li com arguments */noconsole* per a que executi el fitxer en segon pla, i la ruta del propi fitxer a executar.

Al registre de Windows a l'apartat "*HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run* es crea l'entrada com es pot veure a la Fig. 41.

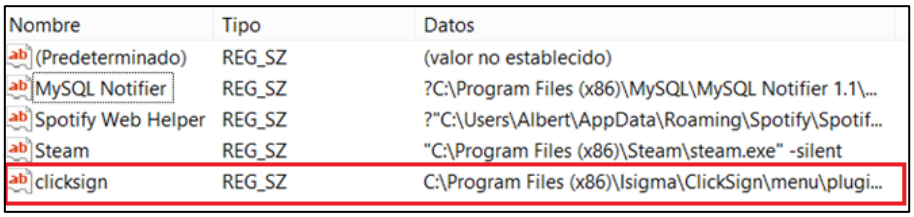

Fig. 41 Resultat de la comanda anterior

3.4.2.3. Nova opció al menú contextual

Com ja s'ha vist anteriorment, CS es pot executar des del menú contextual de Windows. Fent un clic amb el botó dret a un fitxer es pot signar, entrar al menú de configuració, consultar la versió, verificar una signatura, i encendre la funcionalitat SignLoop. Per completar el llistat d'opcions cal afegir la nova funcionalitat del servidor.

El menú contextual s'afegeix a través d'unes DLL (*Dynamic-Link Library*), segons si el sistema operatiu es 32 o 64 bits, aquestes DLL han estat desenvolupades en un projecte annex a l'ISM, i en C++ [40] en comptes de Java. Per tant, caldrà

utilitzar un altre IDE per modificar-les, per exemple Visual Studio 2010 64 bits. Al importar el projecte apareixen cinc carpetes:

- *External Dependències*: conté totes les dependències externes necessàries.
- *Generated Files:* conté els fitxers amb els diferents IID (Interface ID) i CLSID (Class ID).
- *Header Files:* conté els fitxers de capçalera, on es declaren els mètodes, funcions, atributs...
- *Resources Files:* conté les imatges de les opcions del menú contextual, junt amb un fitxer d'extensió rc, anomenat de recursos on es declaren les rutes dels fitxers imatge.
- *Source Files:* conté els fitxers amb extensió *cpp* on estan implementades les funcions.

Per realitzar la implementació de la nova opció, s'ha modificat un dels fitxers amb extensió *cpp*, afegint el tractament de la icona perquè tingui la mida adequada al menú contextual i la nova opció amb la crida que farà per executar la classe Java corresponent. El codi es mostra a la Fig. 42.

```
HBITMAP hPluginPs = isVistaOrMore() ?
IconToBitmapPARGB32(hmod,IDI_PLUGIN_PS) : IconToBitmap(hmod,
IDI_PLUGIN_PS);
uID = configureMenuItemInfo(hSubmenu, 6, uID, PLUGIN, hPluginPs);

[...]
case 6:
      {
      _stprintf_s(szParams,L" -cp \"%s;%s\\lib\\*\"  -Xms%sm -Xmx%sm
es.isigma.ism.winmenu.LinkToPortaSigma appcert:;admin;admin”
m_szDllPath,m_szDllPath,m_szJavaMinMem,m_szJavaMaxMem);
      result = ShellExecute(NULL, L"open",
szCmd,szParams,NULL,SW_SHOWNORMAL);
      return S_OK;
      }
break;
```

Fig. 42 Modificació del fitxer amb extensió cpp

També s'ha modificat un dels fitxers de capçaleres on es declaraven les opcions del menú contextual en les tres llengües disponibles, català, castellà i anglès. El codi es mostra a la Fig. 43.

```
#define STRLANGENG {L"Sign",L"Verify",L"SignLoop",
L"Configure...",L"About ClickSign",L"Request
signatures",L"ClickSign",L"Isigma ClickSign",L"Web Plugin"}
#define STRLANGSPA
{L"Firmar",L"Verificar",L"SignLoop",L"Configuración...",L"Acerca de
```

```
ClickSign",L"Solicitar firmas",L"ClickSign",L"Isigma
ClickSign",L"Plugin PS"}
#define STRLANGCAT
{L"Signar",L"Verificar",L"SignLoop",L"Configuració...",L"Quant a
ClickSign",L"Sol·licitar signatures",L"ClickSign",L"Isigma
ClickSign",L"Web Plugin"}
```

Fig. 43 Modificació del fitxer de capçalera

Després d'aquestes modificacions i algunes més, però menys rellevants, s'ha realitzat la compilació tant per 32 com per 64 bits, i s'han substituït les antigues DLLs per les noves. Finalment el resultat obtingut d'aquest procés és el mostrat en la Fig. 44.

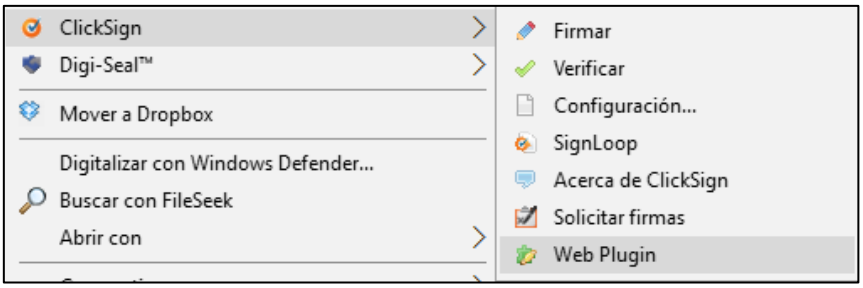

Fig. 44 Opció del *plugin web* afegida

3.4.2.4. Cross-Origin Resource Sharing

Una qüestió a resoldre, ha estat el problema del CORS (*Cross-Origin Resource Sharing*). El CORS apareix quan un recurs realitza una sol·licitud HTTP a un altre recurs d'un domini diferent al que pertany. Per raons de seguretat els navegadors no permeten aquest tipus de sol·licituds iniciades des d'un *script*. Per evitar aquesta situació, s'afegeixen unes capçaleres extres a les crides HTTP i d'igual forma es configura el servidor perquè les permeti.

Això s'ha hagut de fer ja que en el cas de PS i CS, PS realitzava crides cap al domini de *localhost* de CS. Per afegir-les, s'han modificat totes les crides AJAX [41] cap a CS amb la nova capçalera (Fig. 45) i el tractament dels paquets al servidor (Fig. 46).

```
function signDataFromClickSign(challenge, base, algorithm, b64Cert, signOption, totalSigns,
        remSigns, signedChallenge){

        jq.ajax({
            type: 'POST',
            async: true,
            url: "https://localhost:9999/signData"+"?challenge="+challenge+"&base="+base+
            "&algorithm="+algorithm+"&b64Cert="+b64Cert+"&signOption="+signOption+
            "&num="+totalSigns+"&remSigns="+remSigns+"&challengecs="+signedChallenge,
            crossDomain: true,
            success: function(data, textStatus, jqxhr) {
                checkChallenge(data);
            }
        });
}
```

Fig. 45 Crida de PS a CS amb el CORS a *true*

```
@Override
public void handle(HttpExchange t) throws IOException {

    String response = "";
    com.sun.net.httpserver.Headers responseHeaders = t.getResponseHeaders();
    responseHeaders.set("Access-Control-Allow-Headers","Access-Control-Allow-Origin");
    responseHeaders.set("Access-Control-Allow-Origin","*");
    responseHeaders.set("Content-Type","text/plain");
```

Fig. 46 Configuració del CORS al servidor de CS

3.4.3. Compatibilitats amb els diferents sistemes operatius

El servidor al ser desenvolupat amb Java, és compatible des la versió 5 de Java fins a la 8, que és l'ultima fins a dia d'avui. Tots els sistemes operatius Windows, permeten l'execució de Java. Actualment, aquesta solució s'està utilitzant en equips que tenen des de Windows XP fins a W10.
Si que és cert, però, que a Windows XP, el certificat que permet la connexió per SSL no s'autoinstal·la ja que la comanda que s'utilitza per fer-ho, només era compatible a partir de W7. En aquest sistema operatiu, la solució passa per realitzar una crida des del navegador a *localhost:9999* i afegir l'excepció de seguretat.

També, hi ha una excepció, segons la versió de Microsoft Edge que tingui instal·lat el client. Si és la versió amb número de compilació 10158 o superior cal executar un comando des de consola per desactivar aquesta mesura de seguretat que porta Edge, però això ja es realitza en el moment de la instal·lació:

```
ExecShell 'execute' 'CheckNetIsolation LoopbackExempt -a -
n=Microsoft.MicrosoftEdge_8wekyb3d8bbwe'
```

Fig. 47 Comanda NSS per l'execució de crides *localhost* a Microsoft Edge

CONCLUSIONS

Isigma és una empresa especialitzada en la implantació de la signatura electrònica i ofereix diferents aplicacions segons la necessitat del client, com PS per a processos de signatura en el *cloud* o CS per a realitzar signatures des de l'escriptori, ambdues basades en el conjunt de llibreries pròpies ISM.

L'aplicació PS requereix l'ús de Java per tal de poder realitzar totes les operacions criptogràfiques. Arrel de la decisió de Google i d'altres empreses de retirar el suport Java en els seus navegadors, l'empresa ha vist la necessitat de buscar tecnologies alternatives per poder continuar oferint als clients aquestes aplicacions.

En aquest projecte, s'ha realitzat un estudi de diferents possibles tecnologies per substituir l'*applet* utilitzat a PS:

- Complement per navegador web
- Invocació per protocol
- Comunicació per servei
- Invocació per protocol i comunicació per servei
- API totalment Javascript

S'han valorat aquestes solucions tant des del punt de vista del desenvolupador com d'usuari final i s'ha escollit l'opció de comunicació per servei per diferents motius.

- Aprofitament de l'aplicació CS amb les llibreries ISM.
- Alt grau de compatibilitat en tots els sistemes operatius i navegadors web degut al desenvolupament realitzat en Java i JS.
- Presentació de CS a usuaris de PS.
- Rapidesa en el moment de la utilització, ja que el *software* està sempre corrent.
- Procés local sense exposició externa de les dades de signatura.
- No cal servidor extern, ja que la pròpia aplicació és el servidor.

Un cop decidida la nova arquitectura s'ha realitzat la implementació. S'ha començat analitzant el funcionament de PS amb l'*applet*. A continuació s'ha desenvolupat el servidor de CS des de zero, per a que pogués contestar i posteriorment per a que accedís als mètodes criptogràfics de l'ISM i retornés les dades corresponents. Un cop fet això, s'han canviat les crides a l'*applet* per unes crides a CS i s'han tractat els casos en que CS retorna els certificats i els casos en el que CS retorna un codi d'error, en el primer cas per mostrar-los a la taula corresponent i en el segon perquè PS mostri el possible error.

Al tenir PS plenament integrat amb CS, el següent pas ha estat afegir seguretat al sistema. S'ha afegit SSL per xifrar les comunicacions, i s'ha afegit també la triple verificació de la signatura, basada en la validesa de la signatura, en la finestra temporal i en la identificació de PS com a signant del *token*.

A continuació, amb PS i CS ja funcionant i amb un sistema de seguretat afegit, s'ha realitzat tot tipus de modificacions a CS perquè en el moment de la instal·lació, tot quedi correctament configurat. Aquests desenvolupaments han estat:

- Creació i instal·lació des del fitxer d'instal·lació d'un certificat de CA al repositori de confiança del sistema operatiu per permetre la connexió SSL.
- Modificació del fitxer d'instal·lació per modificar el registre de Windows i aconseguir l'execució de CS amb l'inici del sistema operatiu.
- Modificació dels fitxers DLL amb llenguatge C++, per afegir al menú contextual la nova opció *Web Plugin* per encendre el servidor.

Gracies a aquesta solució, PS pot tornar a accedir als certificats instal·lats al sistema operatiu o a un dispositiu criptogràfic connectat a la màquina del client. L'únic inconvenient és la necessitat d'haver d'instal·lar l'aplicació CS, encara que gracies a les mesures de seguretat instaurades, el procés a nivell d'usuari és exactament igual a quan s'utilitzava l'*applet*. Aquesta solució és únicament aplicable a sistemes operatius Windows, però és funcional des de Windows XP fins a l'última versió, Windows 10.

A nivell personal, m'agradaria mencionar que aquest projecte me'l va proposar un amic i company de la carrera. Em va introduir el tema de la criptografia i finalment vaig acceptar realitzar aquest desenvolupament i així aprofitar a documentar-lo com a projecte final de carrera.

El que em va convèncer d'això va ser la possibilitat d'augmentar els meus coneixements en gran mesura, ja que havent cursat el grau de sistemes de telecomunicació no disposava de bons coneixements en aquest àmbit.

En començar a treballar a Isigma, vaig poder veure que l'aplicació web de l'empresa era molt complexa en comparació a les aplicacions realitzades durant els estudis. Es tractava d'una aplicació professional, modular, amb centenars de classes Java, i amb *frameworks* que gestionaven diferents parts de l'aplicació. Gracies a això he aconseguit aprendre molt més sobre Java, sobre diferents *frameworks* com Struts2, Spring, Hibernate... i també tenir coneixements sobre l'eina Maven i el compilador Apache Ant, entre d'altres. A més, m'ha permès aprendre tot un conjunt de metodologies pròpies d'una empresa de software.

El desenvolupament del projecte ha durat 8 mesos aproximadament, ja que primer va ser necessari superar la corba d'aprenentatge mentre s'analitzava quina de les possibles arquitectures implementar, i després realitzar la implementació. A més, a mesura que ens aproximaven als objectius sorgien imprevistos que calia resoldre, com per exemple, un cop vam tenir desenvolupat i testejat tot el sistema al servidor de *preview*, al realitzar la migració al servidor de producció, com aquest portava SSL, les crides AJAX per accedir a CS no es permetien en HTTP, i va caler substituir el servidor HTTP per un HTTPS. O un altre cas va ser, a l'hora de testejar el navegador Microsoft Edge, aquest no acceptava connexions a *localhost* per seguretat. Aquests, són dos dels molts imprevistos que han anat sorgint a mesura que evolucionava l'arquitectura. Tot i

amb això, es va aconseguir implementar la nova tecnologia i, des de febrer de 2016, els clients de Portasigma ja l'utilitzen habitualment.

En resum, la realització d'aquest projecte m'ha permès adquirir uns coneixements en el mon del desenvolupament software que d'altra manera hagués trigat bastant més temps en aconseguir-los i m'ha servit per entrar al mon laboral, i treballar aplicant metodologies reals. I finalment, un cop acabat el desenvolupament del projecte, l'empresa em va proposar seguir amb ells.

Estudi d'ambientalització

La signatura digital implica un gran estalvi d'impressions de documents i, naturalment, això comporta una reducció del consum de tinta i paper. Si s'aconseguís implantar en la majoria dels processos legals de tot el món, el consum de paper amb la seva conseqüent tala d'arbres disminuiria considerablement. A més, tota la documentació impresa físicament durant tants anys, a part de produir despeses del paper, implica disposar de grans magatzems, amb el corresponent manteniment de les instal·lacions. A més, arribarà un moment en el qual ja no serà viable seguir mantenint tanta documentació en paper. En canvi, si tota aquesta documentació es guardés de forma digital, en comptes de necessitar edificis plens d'arxius, tindríem disc durs reutilitzables i que requeririen menys esforços per guardar tanta informació.

BIBLIOGRAFIA

[1] Criptografia de clau pública:
https://ca.wikipedia.org/wiki/Criptografia_de_clau_p%C3%BAblica

[2] Cryptographic Hash Function:
http://www.sans.edu/research/security-laboratory/article/hash-functions

[3] RFC 2315, PKCS#7:
https://www.ietf.org/rfc/rfc2315.txt

[4] RFC 3447, RSA Cryptography Specifications:
http://tools.ietf.org/html/rfc3447

[5] RFC 5246, The Transport Layer Security (TLS) Protocol:
https://www.ietf.org/rfc/rfc5246.txt

[6] PDF Advanced Electronic Signatures, PAdES:
http://www.etsi.org/deliver/etsi_ts/102700_102799/10277801/01.01.01_60/ts_10277801v010101p.pdf

[7] XML Advanced Electronic Signatures, XAdES :
https://www.w3.org/TR/XAdES/

[8] CMS Advanced Electronic Signatures, CAdES:
http://www.rfc-editor.org/rfc/rfc5126.txt

[9] Application Prgroamming Interface:
http://en.wikipedia.org/wiki/Application_programming_interface

[10] Apache ANT:
http://ant.apache.org/

[11] PKCS#12:
https://en.wikipedia.org/wiki/PKCS_12

[12] Online Certificate Status Protocol, OCSP:
https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

[13] Appfuse framework:
http://appfuse.org/display/APF/Home.

[14] Strus 2 with AppFuse:
http://appfuse.org/display/APF/Using+Struts+2

[15] Spring framework:
https://en.wikipedia.org/wiki/Spring_Framework

[16] Hibernate framework:
https://en.wikipedia.org/wiki/Hibernate_(framework)

[17] Apache Maven:
https://maven.apache.org/

[18] CSS:
http://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada.

[19] Netscape Plugin Applicatino Programming Interface, NPAPI:
https://en.wikipedia.org/wiki/NPAPI

[20] Javascript:
http://librosweb.es/libro/javascript/capitulo_1.html

[21] Codification in base 64:
https://en.wikipedia.org/wiki/Base64

[22] Web CryptoKey Discovery:
https://www.w3.org/TR/2013/WD-webcrypto-key-discovery-20130108/

[23] Near Field Communitacion, NFC:
https://en.wikipedia.org/wiki/Near_field_communication.

[24] Personal Computer / Smart Card, PC/SC:
https://es.wikipedia.org/wiki/PC/SC

[25] Protocol handler:
https://msdn.microsoft.com/en-us/library/windows/desktop/bb266526(v=vs.85).aspx

[26] Launch4J: http://launch4j.sourceforge.net/

[27] RFC 4122, A Universally Unique IDentifier (UUID):
https://www.ietf.org/rfc/rfc4122.txt

[28] 3DES Encryption:
https://www.ietf.org/rfc/rfc1851.txt

[29] PKCS1 Signature:
https://en.wikipedia.org/wiki/PKCS_1

[30] HTTP methods, RFC 2616:
https://www.ietf.org/rfc/rfc2616.txt

[31] POST method:
http://www.w3schools.com/tags/ref_httpmethods.asp

[32] Classe Java HttpsServer:
https://docs.oracle.com/javase/7/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/HttpsServer.html

[33] Creació de contextos al servidor HTTPS:
https://docs.oracle.com/javase/7/docs/api/javax/xml/ws/spi/http/HttpHandler.html

[34] Key and Certificate Management, Keytool:
http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html

[35] Creating a KeyStore:
http://docs.oracle.com/cd/E19509-01/820-3503/ggfen/index.html

[36] Extended key usage for SSL:
http://www.ibm.com/support/knowledgecenter/en/SSKTMJ_8.0.1/com.ibm.help.domino.admin.doc/DOC/H_KEY_USAGE_EXTENSIONS_FOR_INTERNET_CERTIFICATES_1521_OVER.html

[37] NSS technology : http://nsis.sourceforge.net/Docs/Chapter1.html.

[38] Instal·lació del certificat de CA desde CMD:
https://technet.microsoft.com/es-es/library/cc732443(v=ws.10).aspx

[39] Modificació del registre de Windows des de NSIS:
http://nsis.sourceforge.net/Add_uninstall_information_to_Add/Remove_Programs

[40] Fitxers .cpp i .h de C++:
http://www.cplusplus.com/forum/articles/10627/

[41] AJAX:
http://www.w3schools.com/ajax/

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFG: Implementació d'un sistema de signatura electrónica per web alternatiu als applets

TITULACIÓ: Grau en Enginyeria de Sistemes de Telecomunicació

AUTOR: Albert Díaz Casellas

DIRECTOR: Olga León Abarca

DATA: 8 de juliol de 2016

ANNEX - TECNOLOGIES UTILITZADES

Per la realització d'aquest desenvolupament, ha estat necessari utilitzar diferents tecnologies. S'ha tingut que realitzar un gran aprenentatge en aquestes tecnologies per tal de poder realitzar les tasques necessàries.

En aquest apartat, s'explicaran, de forma bàsica, les tecnologies utilitzades durant el transcurs del projecte.

1 Java

Java és un llenguatge de programació publicat l'any 1996 i de propòsits generals, concurrent i orientant a objectes. La intenció que té es permetre als programadors desenvolupar el codi i poder-lo executar en qualsevol dispositiu sense necessitat de ser recompilat.

La seva sintaxis prové dels llenguatges C i C++, però té menys utilitats de baix nivell. El codi, es compilat a una classe Java i permet l'execució en qualsevol JVM (Java Virtual Machine).

Applets
Des de la primera versió de Java, existeix la possibilitat de desenvolupar petites aplicacions anomenades *applets*, que posteriorment poden ser incrustades en una pàgina HTML perquè siguin descarregades i executades pel navegador web. Aquests *applets* s'executen en una JVM que el navegador té configurada com extensió en un context de seguretat restringit per impedir que s'executi codi potencialment maliciós.

JRE/JDK
El JRE és es *software* necessari per executar qualsevol aplicació Java.
També s'ofereix el JDK *(Java Development Kit*), el qual, conté el JRE a més d'eines per desenvolupar, com el compilador de Java, Javadoc per generar documentació o el depurador de codi.

4. Javascript

Javascript (JS) és un llenguatge de programació interpretat, i bàsicament orientat a objectes i dinàmic. Pel fet de ser dinàmic, com en la majoria de llenguatges de *scripting*, el tipus està associat al valor i no a la variable, p.e. una variable X pot valer un numero i més endavant pot ser una cadena de caràcters.

Es va dissenyar amb una sintaxis semblant a C, encara que també adopta convencions del llenguatge Java.

S'utilitza principalment del costat del client, implementat per part del navegador web permet millores en la interfície d'usuari i pàgines web dinàmiques, encara que actualment s'està començant a introduir del costat del servidor.

L'ús més comú de JS és el d'escriure funcions en pàgines HTML dins d'un *script*. Les seves principals funcions solen ser:

- Carregar nou contingut per la pàgina enviar dades a un servidor mitjançant AJAX sense necessitat de recarregar la pàgina.
- Animació dels elements de la pàgina, fer-los desaparèixer, canviar la seva mida.
- Continguts interactius com poden ser jocs senzills o reproducció d'àudio i vídeo.
- Validació dels valors d'entrades en un formulari web per assegurar-se que són acceptats abans d'enviar-los al servidor.
- Transmissió d'informació dels hàbits dels usuaris dins la web. Seguiment dels anuncis.

5. AJAX

Asynchronous JavaScript And XML (AJAX) és una tècnica de desenvolupament web que permet crear aplicacions interactives. Aquestes aplicacions s'executen en el client, és a dir, en el navegador mentre es manté una comunicació asíncrona amb el servidor en segon pla. D'aquesta forma es pot realitzar canvis sobre les pàgines web sense la necessitat de recarregar-les.

Un dels possibles inconvenients a l'hora d'utilitzar AJAX, és el CORS, el qual, no permet demanar un recurs d'un altre domini per motius de seguretat, però això es pot solucionar tractant les capçaleres del paquets HTTP tant en el moment de la crida com en el servidor.

6. C++

C++ és un llenguatge de programació dissenyat a mitjans dels anys 80, es va dissenyar com a successor del llenguatge C, i abasta tres paradigmes:

- Programació estructurada. Només s'escriuen funcions que processen dades. Sense estructures de dades complexes com són els objectes, els quals tenen propietats i mètodes propis.
- Programació genèrica. Permet barrejar codi tradicional en C amb codi exclusiu de programació orientada a objectes de C++.
- Programació orientada a objectes. Es defineixen objectes amb certes propietats i funcions, per després envia'ls-hi sol·licituds per a que realitzin els mètodes i canviïn les seves propietats.

Esta ideat per poder treballar tant a alt com a baix nivell, i per això, es considera com un dels llenguatges més potents.

7. NSIS

NSIS és un sistema de codi lliure professional per crear instal·ladors de Windows. Permet instal·lar, desinstal·lar, configurar variables del sistema, descomprimir fitxers, etc.

NSIS es basa en un llenguatge propi de *scripts* per realitzar les tasques. Aquest llenguatge permet afegir lògica al instal·lador de forma que pugui per exemple, comprovar la versió del teu sistema operatiu i optar per una instal·lació o un altre, permetre l'elecció de diferents llengües, comprovar la versió de la pròpia aplicació si ja ha està instal·lada prèviament...

És compatible amb la majoria de les versions de Windows, des de Windows 95 fins Windows 10.

Tota la configuració de l'instal·lador es realitza des del fitxer *installer.nsi*. Aquest fitxer conté tota la lògica que executarà l'instal·lador. En el cas de CS, aquest fitxer està configurat per a:

- Comprovar la versió de Java.
- Comprovar el sistema operatiu.
- Comprovar la versió instal·lada de CS.
- Instal·lar el certificat pel servidor SSL.
- Crear els accessos directes al menú de Windows.
- Crear el desinstal·lador.
- Instal·lar la DLL corresponent a la versió de Windows.
- Modificar el registre de Windows per afegir les opcions al menú contextual.

En la Fig. 48 Fragment del fitxer *installer.nsi*es mostra un fragment del fitxer on es modifica el registre de Windows i on s'atorguen permisos d'administrador a la carpeta arrel d'instal·lació:

```
;Store installation folder
; current version of clicksign dll requires this:
WriteRegStr HKCU "Software\isigma\clicksign" "" $INSTDIR

WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "DisplayName" "${PRODUCT}"
WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "UninstallString" "$INSTDIR\Uninstall.exe"
WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "Publisher" "${COMPANY}"
WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "DisplayVersion" "${VERSION}"
WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "DisplayIcon" "$INSTDIR\clicksign.ico"
WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" \
                 "URLInfoAbout" "http://www.clicksignworld.com/es/"
${GetSize} "$INSTDIR" "/S=OK" $0 $1 $2
IntFmt $0 "0x%08X" $0
WriteRegDWORD HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\${VARIANT}" "EstimatedSize" "$0"
;added for Windows start up as a process
WriteRegStr HKCU 'Software\Microsoft\Windows\CurrentVersion\Run' 'clicksign' '"$INSTDIR\menu\hstart.exe"
                 '/noconsole "$INSTDIR\menu\plugin-ps.bat"'
;Adding privileges to INSTDIR of CS
AccessControl::GrantOnFile "$INSTDIR" "Everyone" FullAccess
```

Fig. 48 Fragment del fitxer *installer.nsi*

8. Maven

Apache Maven és un *software* de gestió i comprensió de projectes. Proporciona als programadors la compilació del *framework*. És una eina que pot compilar diferents projectes, fer desplegaments i compartir dependències en grans projectes.

Maven es basa en un POM *(Project Object Model*). Els projectes, dependències i compilacions de Maven són objectes que poden ser modelats i descrits. Aquests objectes es descriuen a partir del fitxer XML anomenat POM. Aquest esclareix l'ordre de compilació, i quins projectes han de ser compilats, i amb quines dependències. A la Fig. 49 es pot observar els quatre blocs en que es divideix el POM:

- *POM Relationships:* Agrupa les dependències d'altres projectes i llibreries externes amb la versió corresponent que necessitarà el projecte. Hereta les característiques del POM de projectes pare, defineix els seus i pot incloure mòduls secundaris.
- *Build Settings:* Aquest apartat modifica les preferències de compilació del projecte. Es pot canviar la ubicació dels tests i afegir *plugins*.
- *General Project Information:* Informació sobre el projecte com, el nom, els desenvolupadors, llicencies...
- *Build Environment:* Es poden definir els entorns de compilació, amb les corresponents configuracions, p.e. es poden definir dos perfils, un per l'entorn de producció i un pel servidor de *testing,* cadascun amb rutes de fitxers externs i noms de base de dades diferents.

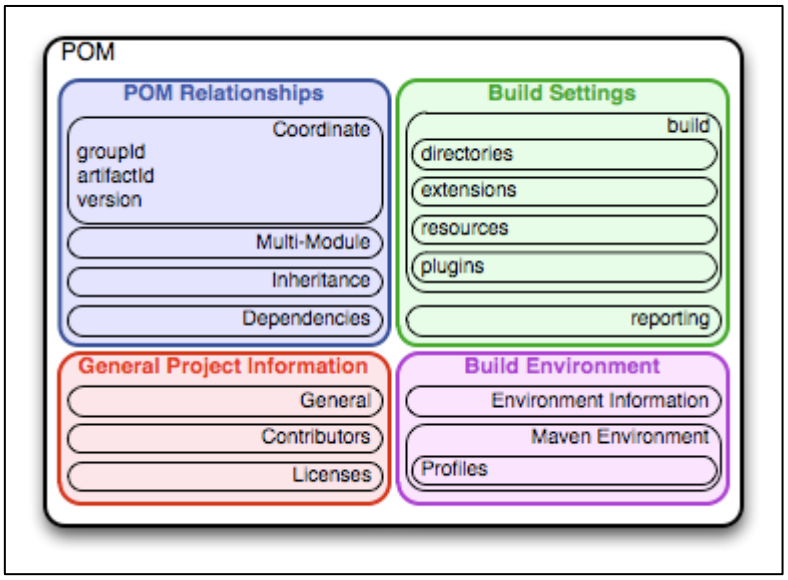

Fig. 49 Blocs que componen un fitxer pom.xml

9. Struts2

Struts2 és un *framework* dissenyat per crear aplicacions web amb Java. Aconsegueix minimitzar el temps de desenvolupament en gran mesura, i ajuda a realitzar el manteniment. Està basat en el patró de disseny model-vista-controlador, en la Fig. 50 es pot veure el funcionament d'aquest patró:

- El model són les dades de l'aplicació.
- La vista són les pròpies plantilles HTML amb les que l'usuari pot interactuar.
- El controlador és l'intermediari entre la vista i el model. El client genera una petició en la vista, i el controlador s'encarrega de modificar el model per a que la vista com a resposta pugui mostrar-li les dades actualitzades.

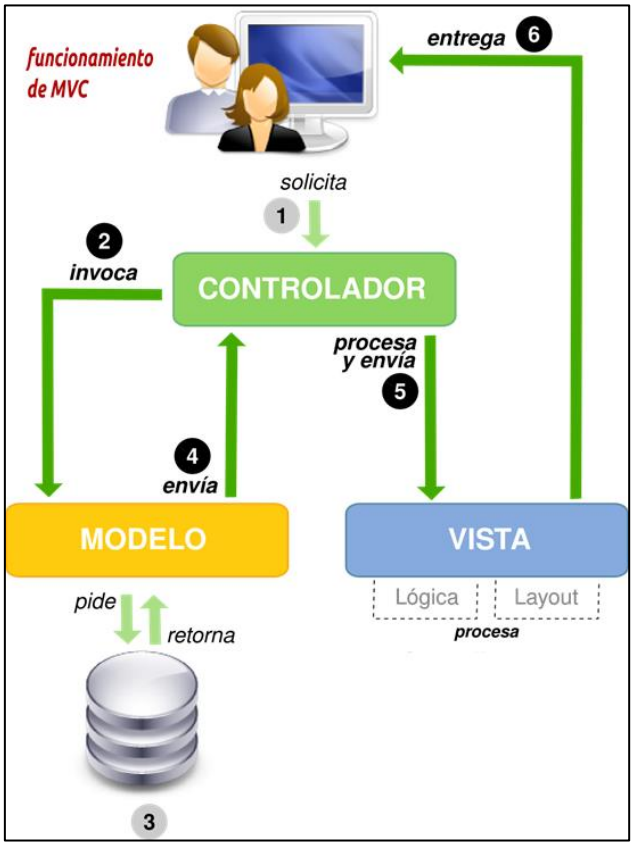

Fig. 50 Esquema del patró MVC

Les seves principals característiques són:

- Suport d'etiquetes Struts
- Integració amb crides AJAX
- Utilització de *plugins*
- Suport per a la generació de vistes a través de *templates*

10. Spring

Spring és un *framework* utilitzat per al disseny d'aplicacions en llenguatge Java. Consta de diferents mòduls que proveeixen un conjunt de serveis a continuació es citaran els més rellevants per al desenvolupament del projecte:

- Inversió de control. Permet evitar lligams entre classes a l'hora de desenvolupar grans aplicacions.
- Autenticació i autorització. Conté processos de seguretat configurables que proporcionen una especial seguretat contra atacs en l'autenticació.
- *Testing*. Conté suport de classes per desenvolupar tests unitaris i d'integració.

També conté un mòdul per a treballar amb el patró model-vista-controlador, però a PS, ja s'utilitza el que té Struts2.

11. Hibernate

Hibernate és una *framework* que s'encarrega del mapeig relacional d'objectes dissenyada per a Java. Amb l'ús d'Hibernate el desenvolupador es treu una part de feina referent a les creacions de base de dades i classes relacionades.

Pel fet d'usar-lo s'obtenen uns avantatges clars:

- Gestió del mapeig entre les classes Java i les taules de la base de dades, gracies a fitxers XML.
- Proveeix d'una API simple per treballar amb els objectes Java directament des de la base de dades.
- Simplicitat de les consultes a la base de dades
- Suporta diferents tipus de base de dades.

En la Fig. 51 es mostra l'arquitectura del *framework.* Com es pot veure només cal configurar el fitxer *hibernate.properties* i *el XML Mapping* encarregat del mapeig.

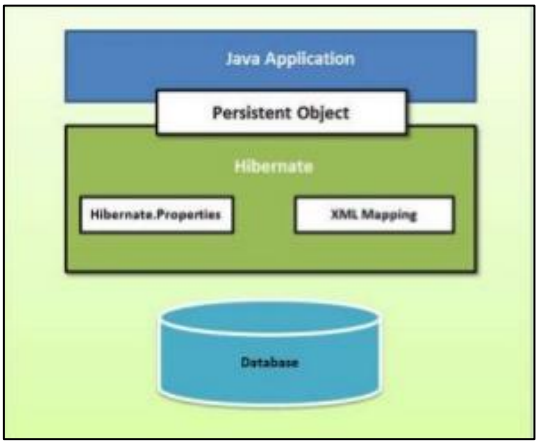

Fig. 51 Arquitectura d'Hibernate

12. Apache Tomcat

Abans d'explicar el propi Tomcat, s'explicarà el que és un *servlet.* Un *servlet* és una classe en Java utilitzada per ampliar les capacitats d'un servidor. Encara que els *servlets* poden respondre a qualsevol tipus de sol·licituds, aquests són utilitzats per estendre les aplicacions allotjades per servidors web. L'ús més comú que tenen, es generar pàgines web de forma dinàmica a partir dels paràmetres de la petició que s'enviï des del navegador web.

Un cop definit, ara s'explicarà el que és un Tomcat.

Apache Tomcat funciona com un contenidor de *servlets*. Implementa les especificacions dels *servlets*. En el cas d'aquest projecte, Tomcat serveix per carregar l'aplicació web PS i que a través del navegador puguem accedir-hi.

13. Java Keystore

Un JKS o *Java Keystore* és un repositori de certificats i claus creat per a Java. Dins d'una *Keystore* poden haver tants certificats amb el seu parell de claus com es desitgi. Aquest repositori ve protegit per una contrasenya i a l'hora els certificats de dins també poden contenir una contrasenya diferent per exportar-los a l'exterior de la JKS. Els JKS solen ser usats pel protocol d'encriptació SSL.

14. Java KeyTool

KeyTool és un eina disponible en els JRE i JDK de Java que permet crear certificats, claus públiques i privades. És configurable, i permet crear el certificat dins d'una JKS o dins d'un altre format com per exemple, PKCS12. A través de comandes permet crear un certificat amb tots els possibles usos, tant de CA autosignat, com un certificat signat per una pròpia CA creada per tu mateix.